Study-Guide: Administrative Tätigkeiten

In diesem Abschnitt werden verschiedene Tätigkeiten der Systemverwaltung behandelt, die ansonsten nicht in einen anderen Abschnitt passen. Verwalten von Benutzer- und Gruppenkonten und verwandte Systemdateien Einstellen von Benutzer- und Systemumgebungsvariablen Konfigurieren und Nutzen der Systemlogs im Rahmen der administrativen und Sicherheitsanforderungen Automatisieren von Systemadministrationsaufgaben durch Planen von zukünftig laufenden Jobs Aufrechterhaltung einer effektiven Datensicherungsstrategie Verwalten der Systemzeit

Seite:	-= LinuxLernSystem =- (http://www.lpi-test.de)
Kurs:	LPIC-1 [102]
Buch:	Study-Guide: Administrative Tätigkeiten
Gedruckt von:	André Scholz
Datum:	Dienstag, 1 November 2005, 10:47 Uhr

Inhaltsverzeichnis

1.111 - Administrative Tätigkeiten

In diesem Abschnitt werden verschiedene Tätigkeiten der Systemverwaltung behandelt, die ansonsten nicht in einen anderen Abschnitt passen.

- Verwalten von Benutzer- und Gruppenkonten und verwandte Systemdateien
- Einstellen von Benutzer- und Systemumgebungsvariablen
- Konfigurieren und Nutzen der Systemlogs im Rahmen der administrativen und Sicherheitsanforderungen
- Automatisieren von Systemadministrationsaufgaben durch Planen von zukünftig laufenden Jobs
- Aufrechterhaltung einer effektiven Datensicherungsstrategie
- Verwalten der Systemzeit

1.111.1 - Verwalten von Benutzer- und Gruppenkonten und verwandten Systemdateien

Beschreibung: Prüfungskandidaten sollten in der Lage sein, Benutzerkonten hinzuzufügen, zu löschen, zu deaktivieren und zu ändern. Enthaltene Tätigkeiten beinhalten das Hinzufügen und Löschen von Gruppen und das Ändern der Benutzer- und Gruppeninformation in den passwd/group Datenbanken. Ebenfalls enthalten ist das Erstellen von speziellen und eingeschränkten Konten.

Die wichtigsten Dateien, Bezeichnungen und Anwendungen:

- **chage**
- **gpasswd**
- **groupadd**
- **groupdel**
- **groupmod**
- **grpconv**
- **grpunconv**
- **passwd**
- **pwconv**
- **pwunconv**
- **useradd**
- **userdel**
- **usermod**
- `/etc/passwd`
- `/etc/shadow`
- `/etc/group`
- `/etc/gshadow`

Um mit einem Linux-Rechner arbeiten zu können, muß man eine gültige Userkennung auf diesem Rechner besitzen. Ein wichtiger Teil der administrativen Aufgaben eines Systemverwalters bezieht sich auf die Verwaltung dieser User-Accounts. Dieser Abschnitt behandelt die dazu notwendigen Fähigkeiten.

Ein User auf einem Linux-Rechner wird intern über seine UserID (UID), eine Nummer von 0 bis 65535, verwaltet. Der User mit der Nummer 0 ist der Systemverwalter. Nicht der Name root sondern die UserID 0 machen ihn zum Superuser. Alle anderen User sind sogenannte Normaluser, deren Rechte zwar stark unterschiedlich sein können, wobei der Unterschied aber nicht über die UserID definiert ist.

Jeder User gehört mindestens einer Gruppe zu, kann aber Mitglied beliebig vieler Gruppen sein. Eine dieser Gruppen ist die Login-Gruppe (manchmal auch primäre Gruppe genannt) des Users, die Gruppe, der später die Dateien gehören werden, die der User anlegt.

Die Dateien, in denen die Informationen über User und Gruppen abgelegt werden und die Befehle, die wir zum Anlegen und Bearbeiten dieser Userinformationen brauchen werden im Folgenden dargestellt.

Die Datei /etc/passwd

Die Datei `/etc/passwd` enthält alle Usereinstellungen, bis auf das verschlüsselte Passwort. In älteren Versionen ist auch das Passwort hier abgelegt, aus Sicherheitsgründen wurde es aber in den neueren Versionen in die Datei `/etc/shadow` ausgelagert.

Die Datei `/etc/passwd` muss für alle User lesbar sein, denn sie ist der einzige Ort, an dem die Verbindung zwischen UserID und Usernamen hergestellt werden kann.

```
-rw-r--r--   ...    /etc/passwd
```

Das wird praktisch bei jedem Aufruf von Programmen benötigt, die auch nur irgendetwas mit Usernamen zu tun haben, sogar ein `ls -l` braucht also diesen Zugriff. Wenn das verschlüsselte Passwort hier abgelegt wäre, so könnte ein Angreifer die Datei lesen, das Passwort nehmen und über eine *brute force attack* (Ausprobieren aller denkbaren Möglichkeiten) das Passwort entschlüsseln. Das war der Grund für die Auslagerung.

Die Datei `/etc/passwd` ist eine Textdatei, die für jeden User eine Zeile angelegt hat. Diese Zeile enthält verschiedene Felder, die durch einen Doppelpunkt voneinander getrennt sind. Das Format ist:

```
Username:x:UserID:GruppenID:Beschreibung:Homeverzeichnis:Startshell
```

Dabei bedeuten die einzelnen Felder:

Username
Der Name des Users. Alphanumerische Zeichen.

x
Hier stand früher das verschlüsselte Passwort. Das x bedeutet, daß die Passwort-Information jetzt in /etc/shadow ausgelagert wurde.

UserID
Die numerische UserID (UID). Eine einheitliche Nummer von 0 bis 65534. Wenn mehrere User die selbe ID benutzen, dann sind sie aus der Sich des Systems die selben User. Die UID 0 ist für den Systemverwalter reserviert.

GruppenID
Die numerische GruppenID (GID) der Login-Gruppe des Users. Der User kann noch Mitglied anderer Gruppen sein, diese Gruppe ist aber seine initiale Gruppe, d.h. alle Dateien, die der User anlegt werden normalerweise die Gruppenzugehörigkeit zu dieser Gruppe haben. (Ausnahme: Dateien, die in Verzeichnissen angelegt werden, die eine andere Gruppenzugehörigkeit haben und das SGID-Bit gesetzt haben)

Beschreibung
Hier kann eine kurze Beschreibung des Users stehen. Das Format ist technisch gesehen frei wählbar. Inzwischen hat sich ein Standard durchgesetzt, der die folgenden Informationen hier ablegt:

- Ausgeschriebener Name des Users (Vor- und Zuname)
- Büro-Raumnummer
- Telephonnummer Büro
- Telephonnummer Privat
- Sonstiges

Diese Informationen werden mit Kommata voneinander getrennt. Programme wie `finger` greifen auf diese Information zu. Dieses Feld kann aber auch leergelassen werden oder einfach nur eine Kurzbeschreibung beinhalten.

Homeverzeichnis
Das Verzeichnis des Users. Beim Login wird automatisch in dieses Verzeichnis gewechselt.

Startshell
Die Shell, die beim Login gestartet werden soll. Hier wird bei normalen Usern meist die /bin/bash oder /usr/bin/csh stehen. Ein Sonderfall sind Useraccounts, die nötig sind um bestimmte Dienste anzubieten (z.B. Samba, pop3, ...). Solche User sollen evt. keine Möglichkeit haben, sich regulär einzuloggen. In einem solchen Fall wird hier zweckmäßigerweise /bin/false eingetragen. Das ist ein Programm, das nichts tut, als einen Rückgabewert von 1 zu produzieren. Wenn ein User diesen Eintrag hat, kann er sich nicht einloggen, bzw. er kann schon, fliegt aber sofort wieder raus, weil seine "Shell" sich sofort wieder verabschiedet.

Eine typische Zeile aus /etc/passwd könnte also folgendermaßen aussehen:

```
root:x:0:0:Hans Maier,1,089/12345,089/54321,Systemverwalter:/root:/bin/bash
```

Oft wird das Beschreibungsfeld gar nicht ausgefüllt oder nur mit einem kurzen Namen versehen, dann wird es noch

einfacher:

```
root:x:0:0::/root:/bin/bash
```

Wenn ein Feld also leergelassen wird, folgen die Doppelpunkte direkt aufeinander.

Die Datei /etc/shadow

Nachdem das verschlüsselte Passwort jetzt nicht mehr in der Datei /etc/passwd abgelegt wird, ist dazu eine andere Datei notwendig. Diese Datei heißt /etc/shadow und birgt neben dem Passwort noch einige Einstellmöglichkeiten, die sich alle um das Passwort drehen. Wie schon in /etc/passwd handelt es sich um eine reine Textdatei, die für jeden User eine Zeile veranschlagt. Wieder ist jede Zeile in Felder aufgeteilt, die durch Doppelpunkte getrennt werden.

Die Datei /etc/passwd ist **nicht lesbar für alle** sondern meist nur durch den Systemverwalter. Manche Distributionen legen eine Gruppe `shadow` an, deren Mitglieder noch ein Leserecht auf diese Datei haben, das ist aber nicht zwingend erforderlich. Ein Zugriffsrecht von

```
-rw-------    ...   /etc/shadow
```

ist völlig ausreichend.

Die interne Struktur jeder Zeile ist folgende:

```
Username:Passwort:Alter:min_Alter:max_Alter:Warnzeit:Pufferzeit:Ungültigkeit:
Reserviert
```

Die Bedeutungen der einzelnen Felder sind:

Username
: Der Username, genau so wie in /etc/passwd. Das ist sozusagen das Schlüsselfeld des Datensatzes, mit dem die Verbindung zu den Zeilen aus /etc/passwd hergestellt wird.

Passwort
: Hier steht das verschlüsselte Passwort. Linux verschlüsselt seine Passwörter mit einem nicht-reversiblen Verschlüsselungsmechanismus durch den Systemaufruf `crypt`. Das Ergebnis ist immer eine 13-24-stellige Zeichenkette, egal wie lange das eingegebene Passwort war. Wenn hier also eine Zeichenkette steht, die nicht 13-24-stellig ist, so existiert kein Passwort, das durch Aufruf von crypt in diese Zeichenkette verschlüsselt werden würde. Damit ist der Eingang faktisch gesperrt. Häufig steht hier z.B. ein einzelnes Sternchen (*) oder ein Ausrufungszeichen (!). Das heißt, daß dieser Eingang verschlossen ist, es existiert kein gültiges Passwort. Der Systemverwalter kann sich aber mit dem Befehl **su** in diesen Benutzer verwandeln und unter seiner Effektiven UID arbeiten. Steht in diesem Feld nichts, folgt also ein Doppelpunkt dem nächsten, so gibt es kein Passwort, der Eingang ist frei, der User kann sich ohne Nennung eines Passworts einloggen.

Alter
: Hier stehen die Anzahl der Tage seit dem 1. Januar 1970 bis zu dem Tag, an dem das Passwort das letzte Mal verändert wurde. Keine Angst, das müssen Sie nicht selbst ausrechnen, das Programm passwd tut das für Sie und trägt diese Zahl hier ein.

min_Alter
: Das Mindestalter eines Passworts, bevor es geändert werden darf. Also genau genommen, die Anzahl der Tage, die seit dem letzten Passwort-Wechsel vergehen müssen, damit es erneut verändert werden darf. Das hat in der Praxis eigentlich nur dann eine Bedeutung, wenn einem User verboten werden soll, das Passwort zu ändern (z.B. wenn mehrere User sich einen Account teilen). Wenn dieses Feld eine größere Zahl aufweist, als das nächste, dann darf der User sein Passwort nie ändern.

max_Alter
Die maximale Anzahl von Tagen, bevor ein Passwort geändert werden muß. Viele Hochsicherheitssysteme verlangen, daß die Passwörter regelmäßig geändert werden müssen. Hier wird die Anzahl der Tage eingetragen, die maximal zwischen zwei Wechseln des Passworts vergehen dürfen.

Warnzeit
Damit ein User merkt, das das Passwort bald geändert werden muß, kann hier die Anzahl der Tage vor dem Ablauf der Gültigkeit eines Passworts angegeben werden, ab der der User beim Login auf den baldigen Ablauf hingewiesen wird.

Pufferzeit
Weil es sein kann, daß ein User ein paar Tage nicht eingeloggt war, kann er womöglich die Warnmeldung nie gesehen haben, die ihn auf das baldige Ablaufen des Passworts hingewiesen hätte. Hier kann man die Pufferzeit in Tagen angeben, die nach dem Ablaufen des Passworts verstreichen darf, bevor der Account tatsächlich ungültig wird.

Ungültigkeit
Dieses Feld wird nur für Accounts benötigt, die zeitlich befristet sind. Hier steht die Anzahl von Tagen seit dem 1. Januar 1970, bis zu dem Tag, an dem der Account ungültig wird. Auch dazu gibt es selbstverständlich Programme, die diesen Wert in ein Datum umrechnen.

Reserviert
Ein reserviertes Feld für zukünftige Erweiterungen.

Ein minimaler Eintrag besteht nur aus dem Usernamen, gefolgt von 8 Doppelpunkten. Das wäre dann ein User, der sich ohne Passwort einloggen darf. Die anderen Felder sind optional und können - je nach Sicherheitsbedarf des Systems - gesetzt werden, wie gewünscht.

Die Datei /etc/group

Diese Datei enthält die Gruppeninformationen für die Usergruppen des Systems. Sie ist - wie die anderen Dateien der Userverwaltung - wieder eine Textdatenbank mit Feldern, die durch Doppelpunkte voneinander getrennt sind. Jede Zeile beschreibt eine Gruppe.

Die Zeilen definieren die Mitglieder der Gruppen, die **nicht** die Login Gruppen der User sind. Die Login-Gruppen-Mitgliedschaft ist in dieser Datei also **nicht** verzeichnet.

Wie schon bei der Datei /etc/passwd, so gilt auch bei der Datei /etc/group, daß sie für alle Welt lesbar sein muß. Daher sind wieder die Passwörter der Gruppen in die Datei /etc/gshadow ausgelagert worden, deren Format gleich im Anschluß besprochen wird. Wie in /etc/passwd, so steht im Passwort-Feld in /etc/group heute einfach ein x.

Das Format der Zeilen in /etc/group ist:

```
Gruppenname:x:GruppenID:Liste der Mitglieder
```

Die Bedeutung der einzelnen Zeilen ist schnell erklärt:

Gruppenname
Der Name der Gruppe, wie er dann z.B. von ls -l ausgegeben wird.

x
Das Feld, in dem früher das Passwort stand.

GruppenID
Die numerische GruppenID (GID), die die Gruppe kennzeichnet. Die Datei /etc/passwd referenziert die Login-Gruppen der Usereinträge z.B. mit dieser Nummer.

Liste der Mitglieder

Eine durch Kommata getrennte Liste von Usernamen. User, die hier aufgeführt sind, sind durch diesen Eintrag Mitglieder der jeweiligen Gruppe. Die Gruppenmitgliedschaft der Login-Gruppe eines Users muß hier nicht aufgeführt sein.

Gruppenmitgliedschaften sind oft nicht ganz richtig verstanden. Aus diesem Grund folgt hier noch eine kurze Beschreibung der Mechanismen:

Jeder User kann Mitglied beliebig vieler Gruppen sein, hat aber immer genau eine Login-Gruppenmitgliedschaft. Diese Login-Gruppenmitgliedschaft wird in /etc/passwd definiert. Jede weitere Gruppenmitgliedschaft eines Users wird in der Datei /etc/group definiert.

Die pure Mitgliedschaft in einer Gruppe gibt dem User die Rechte eines Gruppenmitglieds auf Dateien, die dieser Gruppe zugehören. Es sind keine weiteren Befehle nötig, um diese Rechte wahrzunehmen. Ein Beispiel:

Wenn der User `foo` Mitglied der Gruppen `users, bar, foobar` und `uucp` ist, dann kann er alle folgenden Dateien lesen:

```
-rw-r-----   1 root     users          294 Jul 29  2000 Datei1
-rw-r-----   1 root     bar            294 Jul 29  2000 Datei2
-rw-rw----   1 root     foobar         294 Jul 29  2000 Datei3
-rw-r-----   1 root     uucp           294 Jul 29  2000 Datei4
```

Die Datei `Datei3` dürfte er sogar beschreiben.

Um diese Rechte auszuüben, muß der User `foo` keinerlei zusätzliche Befehle wie newgrp oder ähnliches benutzen, er hat die Rechte einfach durch die bloße Gruppenmitgliedschaft.

Wenn ein User eine Datei anlegt, dann bekommt diese Datei automatisch die Gruppenzugehörigkeit zur Login-Gruppe des Users. Das kann durch Befehle wie newgrp oder sg verändert werden.

Die Datei /etc/gshadow

Gruppenpasswörter werden nicht dazu benötigt, eine Gruppenmitgliedschaft zu beweisen. Sie dienen nur dazu, einem User, der **nicht** Mitglied einer bestimmten Gruppe ist, zeitweilig die Rechte eines Mitglieds zu geben. Dazu muß er das Gruppenpasswort bekommen. Das wird nur sehr selten benötigt, denn es ist ja nicht viel Aufwand, einen User zu einem Gruppenmitglied zu machen.

Auch wenn ein User selbst kein Passwort hat, aber die Gruppe, zu der er mit newgrp wechseln will eines besitzt, muß es angegeben werden.

Damit die Passwörter für Gruppen nicht einlesbar sind, werden sie wieder in einer Datei abgelegt, die nicht für alle Welt lesbar ist. Diese Datei heißt `/etc/gshadow`. Das Format dieser Datei ist wieder ähnlich wie bei den anderen besprochenen Dateien. Zeilen, die in Felder aufgeteilt sind, durch Doppelpunkte voneinander getrennt. Das genaue Format ist:

`Gruppenname``:`*`Passwort`*`:`*`Gruppenverwalter`*`:`*`Liste der Mitglieder`*

Die einzelnen Felder haben folgende Bedeutung:

Gruppenname
Der ausgeschriebene Name der Gruppe, wie er in /etc/group steht

Passwort
Das verschlüsselte Passwort der Gruppe. Steht hier eine Zeichenkette, die kein gültiges verschlüsseltes Passwort darstellt, also eine nicht 13-24 stellige Zeichenkette, dann hat die Gruppe kein gültiges Passwort. Das heißt, User, die nicht Mitglied der Gruppe sind, sich nicht mit newgrp zum Gruppenmitglied erklären können.

Gruppenverwalter

Jede Gruppe kann einen User haben, der als Gruppenverwalter fungiert. Ein Gruppenverwalter darf

- Andere User in die Gruppe aufnehmen oder aus der Gruppe ausschließen.
- Das Gruppenpasswort ändern.
- Das Gruppenpasswort löschen.

In diesem Feld steht der Name des Gruppenverwalters der Gruppe, so wie er auch in /etc/passwd aufgeführt ist.

Liste der Mitglieder

Eine Liste der Mitglieder der Gruppe, die sich mit newgrp zum Mitglied der Gruppe machen dürfen. Diese Liste ist nicht zwansläufig deckungsgleich mit der der gleichen Gruppe in /etc/group. Ein User, der Mitglied hier in der Liste ist, aber nicht als Mitglied der Gruppe in /etc/shadow aufgeführt ist, ist nicht automatisch Mitglied der Gruppe, kann sich aber ohne Nennung eines Passworts mit newgrp zum Mitglied der Gruppe ernennen.

Der wichtigste Grund für die Verwendung der /etc/gshadow-Datei ist wohl der, daß jede Gruppe einen eigenen Gruppenverwalter haben kann. Diese Tatsache erleichtert das Leben des Systemverwalters, weil er sich nicht um alles selbst kümmern muß.

Programme zum Umgang mit den User-/Gruppendatenbanken

Die Arbeiten an all diesen vier Dateien (passwd, shadow, group und gshadow) können prinzipiell immer mit einem Texteditor wie dem vi vorgenommen werden, es gibt aber auch viele wichtige Programme, die einem Systemverwalter die zeitaufwendige Handarbeit ersparen und auch in Schleifen bestens dazu geeignet sind, Veränderungen für viele User (oder Gruppen) automatisch vorzunehmen. Dazu kommt, daß es Programme geben muß, mit deren Hilfe auch Normaluser ihre Einstellungen verändern können, etwa ihr Passwort wechseln. Diese Programme sollen hier noch kurz dargestellt werden. Ich habe - wo immer es noch keine deutsche Handbuchseite gab - die jeweiligen Handbuchseiten übersetzt und hier mit aufgenommen.

Die beste Art, diese Programme kennenzulernen, ist es, möglichst viel mit ihnen herumzuspielen, neue User anzulegen, zu verändern, wieder zu löschen usw. Nur die Praxis mit Programmen führt im Endeffekt zur Vertrautheit mit dem Umgang...

Das Anlegen neuer User mit useradd

Das Programm useradd dient dazu, einen neuen User anzulegen, ohne die Handarbeit mit Editoren auf den Dateien passwd, shadow, group und gshadow. Wir haben oben schon gesehen, daß das Format z.B. der Datumsangaben (Tage seit dem 1. Januar 1970) nicht unbedingt geeignet ist, von Hand gesetzt zu werden. Das Programm **useradd** nimmt uns diese Arbeit ab.

Die genaue Anwendung dieses Programms entnehmen Sie der Handbuchseite, hier nur noch ein paar Tipps aus der Praxis:

Wenn mit useradd ein neuer User angelegt werden soll, dann reicht oft nur die Angabe des Namens, weil die Voreinstellungen in der Regel sehr vernünftige Werte beinhalten. Oft wird jedoch vergessen, daß das Programm ohne die Option **-m** das Homeverzeichnis des Users nicht anlegt. Daran sollten Sie denken.

Die Angabe eines Passwortes ist selten von Nöten. Wird useradd ohne Passwort angegeben, so sperrt es den Account zunächst einmal durch ein ! im Passwortfeld der Datei /etc/shadow. Später können Sie mit dem Programm passwd ein Passwort vergeben.

Löschen von Usereinträgen mit userdel

Das Löschen von Useraccounts übernimmt das Programm userdel. Wird zusätzlich noch die Option **-r** angegeben, so wird gleich das Homeverzeichnis des Users mitgelöscht. Andere Dateien im System, die diesem User gehören werden nicht gesucht. Hier muß "von Hand" gesucht werden, das nimmt uns **userdel** nicht ab. Aber wir kennen ja

jetzt das Programm find, mit dem wir solche Suchen in einem Rutsch erledigen können...

Ändern von Usereinstellungen mit usermod

Um die ganzen Eigenschaften eines bereits bestehenden Users zu verändern gibt es das Programm usermod. Hier können die ganzen Einstellungen verändert werden, die wir schon beim Anlegen eines Users angeben konnten. Zusätzlich kann auch der Name des Users noch verändert werden.

Bei der Änderung verschiedener Eigenschaften kommt es aber immer wieder zu Komplikationen, ein paar davon seien hier genannt:

- **Ändern der UserID**
  Wird die UserID eines Users verändert, dann stimmen die Rechte des Users nicht mehr. Denn die Rechte des Users auf Dateien werden ja über den Eintrag der UID in der Inode bestimmt. Aus diesem Grund verändert **usermod** zwar die UserID aller Dateien und Verzeichnisse innerhalb des Homeverzeichnisses, jedoch nicht die der restlichen Dateien im System. Das muß wiederum mit find erledigt werden.
- **Ändern des Usernamens**
  Wird der Username geändert, so verändert sich damit nicht automatisch der Name des Homeverzeichnisses. Das kann zu Inkonsistenzen führen, wenn es die Regel ist, daß alle User ihre Homeverzeichnisse in `/home/`*Username* vorfinden, nur der User Otto hat seins in /home/maier... Es ist kein technisches Problem, aber das Leben wird einfacher, wenn man sich an Prinzipien hält.
- **Ändern des Homeverzeichnisses**
  Wenn also auch das Homeverzeichnis geändert werden soll, so darf nicht vergessen werden, die Option **-m** mit anzugeben, sonst werden die Dateien des alten Homeverzeichnisses nicht ins neue verschoben bzw. das neue wird gar nicht erst angelegt.

Ändern der Passwort-Gültigkeit mit chage

Speziell für die Einträge in `/etc/shadow` existiert noch das Programm chage, das entweder über Kommandozeilenparameter oder - falls keine Parameter angegeben wurden - interaktiv die Einstellungen dieser Werte erlaubt.

Ändern von Usereigenschaften mit passwd, newgrp, chsh und chfn

Alle drei Programme **useradd, userdel** und **usermod** sind Werkzeuge des Systemverwalters und werden nur von ihm aufgerufen. Damit auch Normaluser ein paar Dinge - zumindestens für ihren eigenen Account - ändern dürfen gibt es noch ein paar Programme, deren Aufgaben sich zum Teil mit denen der drei letztgenannten überschneiden - passwd, newgrp, chsh und chfn.

Diese Programme ändern alle (mit Ausnahme von newgrp) verschiedene Einstellungen in einer oder mehreren der vier oben beschriebenen Dateien passwd, shadow, group und gshadow. Die genaue Anwendung entnehmen Sie wiederum den Handbuchseiten. Hier eine kurze Beschreibung, was die einzelnen Programme anbieten und was Normaluser damit anstellen können:

- **passwd**
  Mit diesem Programm kann ein Normaluser sein Passwort ändern. Ein Verwalter einer Gruppe kann damit zusätzlich auch das Gruppenpasswort dieser Gruppe ändern, der Systemverwalter kann alle Passwörter aller User und Gruppen ändern.
- **newgrp**
  Mit diesem Kommando kann ein User kurzfristig seine Logingruppe wechseln. Das heißt, daß Dateien, die er anlegt jetzt zu dieser neuen Gruppe gehören, statt zur eigentlichen Logingruppe des Users. Wenn der User selbst nicht Mitglied der gewünschten Gruppe ist, jedoch das Gruppenpasswort kennt, so kann er nach der Eingabe des Gruppenpassworts trotzdem kurzfristig Mitglied dieser Gruppe werden. Dieses Programm ändert nichts an den Systemdateien, beim nächsten Einloggen ist alles so wie vorher.
- **chsh**
  Jeder User kann mit diesem Programm den Eintrag der Startshell in seinem Useraccount in /etc/passwd ändern. Dieses Programm ändert nur diesen Eintrag, erst durch ein Logout mit anschließendem erneuten Login wird die Änderung spürbar. Damit ein User nicht jedes beliebige Programm hier eintragen kann, muß die gewünschte Shell in `/etc/shells` aufgelistet sein. Der Systemverwalter kann mit diesem Programm

jedem User eine andere Startshell zuweisen.
- **chfn**
  Jede Zeile der /etc/passwd enthält ein Kommentarfeld, in dem verschiedene Informationen über einen User eingegeben werden können. Jeder User kann diese Informationen selbst ändern oder löschen (mit Ausnahme eines Feldes). Dazu benutzt er das Programm **chfn** (CHange Full Name). Selbstverständlich kann ein Normaluser damit nur das Feld seiner eigenen Zeile ändern, der Systemverwalter kann aber damit die Einträge aller User editieren.

Die beiden letzten Programme arbeiten auch interaktiv. Wenn keinerlei Parameter angegeben werden, so fragen diese Programme die Angaben ab.

Verwaltung der Gruppen

Die Aufgaben zum Verwalten von Gruppen entsprechen im Wesentlichen denen zur Verwaltung der User. Und so existieren für den Systemverwalter (und nur für ihn) hier genau die drei Befehle, die wir für die Userverwaltung schon hatten:

- **groupadd**
  Zum Anlegen von Gruppen
- **groupmod**
  Zum Ändern der bestehenden Eigenschaften der Gruppen
- **groupdel**
  Zum Löschen von Gruppen.

Mit der Einführung der Shadow-Gruppen, also der verschlüsselten Speicherung der Gruppenpasswörter in der Datei `/etc/gshadow` hielt eine andere Neuerung ihren Einzug, die Möglichkeit, daß bestimmte User zu Gruppenadministratoren werden. Ein Gruppenadministrator hat bestimmte Rechte für die Gruppe, deren Verwalter er ist. Er kann

- User in eine Gruppe als Mitglied aufnehmen
- Mitglieder einer Gruppe von der Gruppenmitgliedschaft ausschließen
- Gruppenpasswörter setzen, verändern und löschen.

Für diese Aufgabe steht ihm das Programm gpasswd zur Verfügung. Dieses Programm ist prinzipiell für zwei Aufgabenbereiche geeignet:

- Der Systemverwalter kann damit den Gruppenverwalter bestimmen.
- Der Gruppenverwalter kann damit die oben genannten Aufgaben erledigen.

Und schließlich gibt es noch das Programm groups, das jeder User dazu nutzen kann, herauszufinden in welchen Gruppen er eigentlich Mitglied ist. Dieses Programm ist eigentlich nur ein Shellscript, das wiederum den Befehl id nutzt. Id ist ein Programm, mit dem herausgefunden werden kann, welche UserID, GruppenIDs usw ein User besitzt.

Überprüfung der Konsistenz mit pwck und grpck

Da all diese Verwaltungsaufgaben für die oben beschriebenen User- und Gruppendateien viel Aufwand ist, gibt es noch die Programme pwck (Password-Check) und grpck (Group-Check), die die interne Konsistenz dieser Dateien überprüfen. Diese Programme sind naturgemäß Werkzeuge des Systemverwalters, da außer ihm niemand die shadow-Dateien lesen darf.

Konvertierung der User- und Gruppendatenbanken

In seltenen Fällen kann es notwendig sein, die Passwort- und Gruppendatenbanken entweder von der alten (shadow-losen) Version in die moderne (shadow-basierte) Version zu konvertieren oder umgekehrt. Zu diesem Zweck stehen uns die folgenden Programme zur Verfügung:

pwconv	Konvertiert die alte /etc/passwd Datei in eine passwd/shadow Kombination.

pwunconv Konvertiert eine moderne passwd/shadow Kombination in eine alte passwd Datei.
grpconv Konvertiert die alte /etc/group Datei in eine group/gshadow Kombination.
grpunconv Konvertiert eine moderne group/gshadow Kombination in eine alte group Datei.

1.111.2 - Einstellen von Benutzer- und Systemumgebungsvariablen

Beschreibung: Prüfungskandidaten sollten in der Lage sein, globale und Benutzerprofile zu modifizieren. Dieses Lernziel beinhaltet das Setzen von Umgebungsvariablen, das Verwalten von skel Verzeichnissen für neue Benutzerkonten und das Setzen des Suchpfades auf die richtigen Verzeichnisse.

Die wichtigsten Dateien, Bezeichnungen und Anwendungen:

- **env**
- **export**
- **set**
- **unset**
- `/etc/profile`
- `/etc/skel`

Profildateien

Wenn sich ein User an einem Linux-System anmeldet, sich also dort mit Usernamen und Passwort einloggt, so wird als erste Aktion, nach dem Login eine Shell gestartet. Welche Shell gestartet werden soll steht in dem jeweiligen Eintrag des Users in `/etc/passwd`. Jede Shell kann beim Start konfiguriert werden, das heißt, sie durchläuft mehrere Konfigurationsdateien, in denen bestimmte Umgebungsvariablen gesetzt werden können und evt. andere Programme aufgerufen werden, die im Hintergrund Arbeiten verrichten, die bei jedem Login erneut ausgeführt werden müssen.

Bei den Startdateien gibt es grundsätzlich zwei verschiedene Kategorien. Zum Einen, die Dateien, die bei jedem Login jedes Users ausgeführt werden und zum Anderen, die userbezogenen Dateien, in denen jeder User seine Einstellungen selbst treffen kann. Die systemweiten Startdateien liegen grundsätzlich in /etc, die userbezogenen grundsätzlich im Home-Verzeichnis des jeweiligen Users.

Linux/Unix kennt verschiedene Standard-Shells, die zum Teil unterschiedliche solche Startdateien benutzen. Im weiteren Verlauf dieser Darstellung werde ich mich auf die der Linux-Standardshell BASH reduzieren. Hier nur einmal kurz eine Auflistung verschiedener Startdateien:

<table>
<tr><th>Art</th><th>Bourne Again Shell</th><th>Korn Shell</th><th>TC-Shell</th></tr>
<tr><td>Systemweit</td><td colspan="2"><code>/etc/profile</code></td><td><code>/etc/csh.cshrc</code>
<code>/etc/csh.login</code></td></tr>
<tr><td rowspan="2">Userbezogen</td><td><code>~/.bash_profile</code>
<code>~/.bash_login</code></td><td></td><td rowspan="2"><code>~/.tcshrc</code> oder <code>~/.cshrc</code>

<code>~/.history</code>
<code>~/.login</code>
<code>~/.cshdirs</code></td></tr>
<tr><td colspan="2"><code>~/.profile</code></td></tr>
</table>

Aus der Tabelle ist zu ersehen, daß die **bash** und die **ksh** zumindestens zwei Dateien gemeinsam benutzen, während die **csh** nur eigene Startdateien benutzt. Das liegt an der Tatsache, daß **bash** und **ksh** scriptkompatibel sind (also die selbe Syntax bei Shellscripts haben) während die **csh** eine völlig andere Syntax für ihre Scripts benutzt.

Startdateien der Bourne Again Shell (bash)

Die Bourne Again Shell benutzt verschiedene Startdateien, die in unterschiedlichen Fällen abgearbeitet werden und in denen verschiedene Aufgaben erledigt werden können. Zunächst muß aber bei der Shell zwischen verschiedenen Kategorien unterschieden werden. Die erste grundsätzliche Unterscheidung liegt zwischen

interaktiver Shell und *nicht interaktiver Shell*. Eine Shell ist immer dann eine interaktive Shell, wenn der User darin Kommandos eingeben kann. Das ist jedesmal der Fall, wenn

- ein User sich neu am System anmeldet,
- ein X-Term geöffnet wird,
- eine Shell eine andere Shell durch Programme wie su aufruft,
- eine Shell durch den User direkt aufgerufen wird.

Eine Shell ist eine nicht interaktive Shell immer dann, wenn z.B. eine Subshell aufgerufen wird, um ein Shellscript abzuarbeiten. Eine nicht-interaktive Shell arbeitet keine Startdateien ab.

Bei den interaktiven Shells muß zusätzlich noch zwischen einer *Login-Shell* und einer *Nicht Login-Shell* unterschieden werden. Die Shell ist dann eine Login-Shell, wenn sie direkt vom Login-Programm beim Anmelden eines Users gestartet wurde, oder wenn sie explizit mit dem Parameter `-l` aufgerufen wurde. Jede andere Shell, also wenn z.B. aus einem XTerm ein anderes Xterm aufgerufen wird, ist eine *nicht Login-Shell*.

Der wesentliche Unterschied zwischen Login-Shell und Nicht-Login-Shell besteht darin, daß bei einer Login-Shell davon ausgegangen werden muß, daß noch keinerlei Umgebungsvariablen gesetzt sind. Hier ist es also nötig, daß der gesammte Konfigurationsvorgang durchgearbeitet werden muß, um sicherzustellen, daß die Umgebung den Anforderungen entspricht. Eine Nicht-Login-Shell hingegen wurde ja durch eine Login-Shell aufgerufen, die ihrerseits wieder den Konfigurationsvorgang durchwandert hat. Ausgehend von der Forderung, daß alle wichtigen Umgebungsvariablen durch die **export**-Anweisung weitervererbt werden, muß diese Shell also nicht nochmal konfiguriert werden. Der Aufruf wird so wesentlich schneller.

Konfigurationsdateien der Loginshell

Eine Login-Shell arbeitet zuallererst immer die Datei **/etc/profile** ab. Diese Datei enthält alle wichtigen Konfigurationen, die für alle User des Systems gültig sein sollen. Hier werden insbesondere die folgenden Variablen gesetzt:

- **PATH**
  Der Suchpfad für ausführbare Dateien. Die Reihenfolge der genannten Verzeichnisse ist wichtig. Das erste gefundene Programm wird ausgeführt.
- **MANPATH**
  Der Pfad, in dem die verschiedenen Handbuch-Hierarchien zu finden sind. Auch hier ist die Reihenfolge wichtig. Liegen etwa die deutschen Handbuchverzeichnisse hier nach den englischen, dann werden sie nur angezeigt, wenn mit `man -a` alle Seiten angefordert werden würden.
- **INFODIR** bzw. **INFOPATH**
  Der Pfad zu den Info-Seiten.
- **PAGER** und **EDITOR**
  Die Standardprogramme zum Ansehen und Editieren von Textdateien (z.B. less und vi).
- **PS1** und **PS2**
  Die Prompt-Einstellungen für den normalen und den Folgeprompt der Shell.

Daneben können hier weitere Variablen wie **LS_OPTIONS** oder **LESSCHARSET** gesetzt werden.

Auch Aliase und Shellfunktionen, die für alle User gelten sollen, werden in dieser Datei definiert. Eines ist aber für alle hier gesetzten Variablen, Funktionen und Aliase wichtig: **Alle hier gesetzten Variablen, Aliase und Funktionen, die auch in später aufgerufenen Shells Geltung haben sollen, müssen mit der Anweisung `export` exportiert werden!**

Absolut wichtig ist in **/etc/profile** die Angabe des initialen **umask**-Kommandos, das eine vernünftige Grundeinstellung für die Zugriffsmodi neu zu erstellender Dateien und Verzeichnisse setzt.

Des weiteren können hier noch Grenzwerte für die User gesetzt werden, was die Verwendung von Rechenzeit und Speicher angeht. Das wird durch Aufrufe des Programms **ulimit** erreicht.

Wenn die Datei /etc/profile abgearbeitet ist, werden noch die userspezifischen Konfigurationsdateien abgearbeitet. Hier stehen drei verschiedene Dateien zur Verfügung, von denen nur eine einzige wirklich abgearbeitet wird. Die

drei Dateien liegen alle im Heimatverzeichnis des jeweiligen Users und heißen **`.bash_profile .bash_login`** und **`.profile`. Die Shell entscheidet nach der folgenden Regel, welche dieser drei Dateien abgearbeitet wird:**

```
if [ -e ~/.bash_profile ]
then
  . ~/.bash_profile
elif [ -e ~/.bash_login ]
then
  . ~/.bash_login
elif [ -e ~/.profile ]
then
  . ~/.profile
fi
```

Oder - für Nicht-Bash-Programmierer:

Wenn im Heimatverzeichnis des Users die Datei .bash_profile existiert wird sie abgearbeitet.

Wenn diese Datei nicht existiert wird die Datei .bash_login im Heimatverzeichnis des Users gesucht und falls gefunden, abgearbeitet.

Wenn auch diese Datei nicht existiert, so wird im Heimatverzeichnis des Users die Datei .profile gesucht und falls gefunden abgearbeitet.

Diese drei Konfigurationsdateien bieten eigentlich jeweils die selben Möglichkeiten, wie die Einstellungen in /etc/profile, mit dem wesentlichen Unterschied, daß diese Einstellungen nur für den jeweiligen User gelten, in dessen Verzeichnis sich die Dateien befinden. Da immer nur eine dieser Dateien abgearbeitet wird, kann hier elegant jongliert werden. Ein paar Beispiele:

Will ein User experimentieren, so kann er alle Experimente in einer der beiden ersten Dateien machen, während er eine gut funktionierende dritte Datei besitzt. Falls etwas schiefgeht, muß er nur die ersten Dateien löschen und es wird wieder die dritte abgearbeitet.

Will ein Systemverwalter seinen Usern verbieten, Experimente oder eigene Einstellungen zu machen, muß er nur die Einstellungen in die erste abzuarbeitende Datei machen und dem User kein Schreibrecht darauf geben.

Neben diesen Startdateien kann auch noch eine Datei mit Namen **.bash_logout** existieren, die dann beim Logout abgearbeitet wird. Hier ist Platz für eventuell anstehende Aufräumarbeiten oder ähnliches.

Die Konfigurationsdatei der Nicht-Loginshell

Auch eine Nicht-Loginshell muß eventuell noch etwas konfiguriert werden. Das ist zwar nicht zwingend der Fall, insbesondere dann nicht, wenn die obigen Konfigurationsdateien der Loginshell alle wichtigen Einstellungen exportieren, aber die Shell kennt die Möglichkeit dazu. Allerdings ist diese Möglichkeit immer nur userbezogen.

Wenn im Verzeichnis eines Users die Datei **.bashrc** existiert, so wird sie von einer Nicht-Loginshell abgearbeitet, sobald diese Shell aufgerufen wird.

Grundsätzliches zu Umgebungsvariablen

Jede Shell und natürlich auch unsere Standard-Shell **bash** kann Variablen definieren. Die Summe aller definierten Variablen nennen wir die Umgebung der Shell (engl. environment).

Shellvariablen werden einfach durch die Angabe

Variablenname=Wert

definiert. Durch diese Definition steht uns die Variable mit dem gewählten Namen in **dieser und nur dieser** Shell

zur Verfügung. Damit eine definierte Variable auch in Subshells zur Verfügung steht, die von unserer Shell geöffnet werden, muß die Variable **exportiert** werden. Das kann entweder im Nachhinein durch die Anweisung

export *Variablenname*

oder schon während der Definition durch

export *Variablenname=Wert*

geschehen. Erst jetzt werden solche Variablen auch in späteren Subshells zur Verfügung stehen. Allerdings stehen sie dort nur als Kopien der Variable der aufrufenden Shell zur Verfügung. Das hat zur Folge, daß die Variablen, die in der Subshell verändert werden, in der aufrufenden Shell unverändert bleiben.

Folgendes Beispiel mag das verdeutlichen:

`NAME=hans`	Eine Variable wird erzeugt
`export NAME`	Sie wird exportiert
`bash`	Eine Subshell wird geöffnet
`echo $NAME`	Wir überprüfen den Export
`hans`	Hat funktioniert
`NAME=otto`	Ein neuer Wert für die Variable
`echo $NAME`	Wir überprüfen den neuen Wert
`otto`	Hat funktioniert
`exit`	Subshell wird geschlossen
`echo $NAME`	Inhalt der Variable in der ersten Shell
`hans`	Ist immer noch der alte Wert.

Jede Shell hat ihren eigenen Umgebungsspeicher, in den zwar jeweils Kopien der exportierten Variablen abgelegt werden, der aber beim Beenden der Shell wieder gelöscht wird. So kann in keinem Fall eine geänderte Variable einer Subshell wieder der ursprünglichen Shell "reimportiert" werden.

Jedes Shellscript, das aufgerufen wird, wird in einer Subshell abgearbeitet. Variablen, die in diesem Script erzeugt oder verändert werden, sind nach Beendigung des Scripts (und damit der ausführenden Subshell) nicht mehr aktuell.

Damit es aber trotzdem möglich ist, Shellscripts zur Definition von Variablen heranzuziehen, gibt es die Möglichkeit ein Script in der ursprünglichen Shell auszuführen, statt eine Subshell zu öffnen. Zu diesem Zweck wird ein einfacher Punkt (.) mit anschließendem Leerzeichen vor der Nennung des Scriptnamens angefügt.

`meinscript`	meinscript wird von einer Subshell ausgeführt
`. meinscript`	meinscript wird von der ursprünglichen Shell ausgeführt.

Durch den Punkt wird unsere Shell also gezwungen, das Script selbst abzuarbeiten. Die Variablen, die innerhalb dieses Scripts definiert wurden, sind jetzt auch nach der Abarbeitung des Scriptes noch gültig.

Das klingt schön, hat aber einen bedeutenden Haken. Wenn innerhalb des Scrips ein `exit` vorkommt, normalerweise dazu benutzt, um das Script zu beenden, wird ja die Shell, die das Script ausführt, beendet. Wurde das Script jetzt mit vorgestelltem Punkt aufgerufen, also von unserer Loginshell selbst, dann wird das exit die LoginShell beenden! Wir müßten uns also erneut einloggen.

Scripts, die mit führendem Punkt aufgerufen werden sollen, sollten daher auf keinen Fall - auch nicht zur Fehlerbehandlung - ein `exit` benutzen.

Eine Liste aller Umgebungsvariablen, die in der aktuellen Shell bekannt sind (zusammen mit ihren Inhalten) wird durch das Shellkommando **set** ausgegeben.

Eine Variable kann durch den Befehl

```
unset Variablenname
```

wieder aus dem Umgebungsspeicher entfernt werden.

Schließlich ist es noch möglich, einen Befehl in einer speziellen Umgebung auszuführen, die wir mit dem Befehl env erzwingen können.

Das Verzeichnis /etc/skel

Wenn ein neuer User angelegt werden soll, dann hätte der Systemverwalter viel Arbeit damit, jedem neu anzulegenden User all diese persönlichen Konfigurationsdateien eigens zu erstellen und einzurichten. Damit diese Arbeit gar nicht erst anfällt, existiert das Verzeichnis **/etc/skel**. Es enthält alle wichtigen Konfigurationsdateien, die in einem Userverzeichnis existieren sollten, damit er vernünftig arbeiten kann.

Beim Anlegen eines neuen Users werden diese Konfigurationsdateien dann in das neu angelegte Userverzeichnis kopiert und dem neuen User übereignet. Das wird automatisch dann ausgeführt, wenn das **useradd**-Kommando mit der Option `-m` aufgerufen wurde.

Ein Systemverwalter kann also sozusagen einen Musteruser anlegen und alle nötigen Konfigurationsdateien dieses Musterusers in dieses Verzeichnis speichern. Jeder neu angelegte User wird dann exakt die Konfiguration des Musterusers bekommen.

Neben den oben besprochenen Konfigurationsdateien der Shells finden sich hier auch noch viele weiteren Dateien, deren Namen praktisch immer mit einem Punkt beginnt und somit von einem normalen `ls` Kommando nicht angezeigt wird. Diese Dateien sind Konfigurationsdateien für verschiedene Programme, die ein Normaluser ausführen können soll. Welche Dateien im Einzelnen hier zu finden sind, muß der Systemverwalter entscheiden. Ein paar Beispiele:

`.Xdefaults`
: Persönliche Resourcen für X-Clients. Hier kann das Verhalten und Aussehen verschiedener X-Clients eingestellt werden.

`.Xresources`
: Persönliche Resourcen für X-Clients. Hier kann das Verhalten und Aussehen verschiedener X-Clients eingestellt werden. Exakt die selbe Funktion wie .Xdefaults.

`.Xmodmap`
: Tastaturbelegung für X11. Spezielle Einstellungen, wie die Tasten belegt werden sollen.

`.bash_history`
: Eine leere Datei zur Aufnahme der bereits eingegebenen Befehle der Shell (Befehlswiederholung).

`.dayplan`
: Termine des Users für das Programm plan

`.dayplan.priv`
: Private Termine des Users, auch für plan

`.emacs`
: Persönliche Konfiguration für den Editor emacs

`.exrc`
: Persönliche Konfiguration für den Editor ex

`.vimrc`
: Persönliche Konfiguration für den Editor vim

`.gimprc`
: Persönliche Konfiguration für das Bildbearbeitungsprogramm gimp

`.xinitrc`
: Persönliche Konfiguration des Starts von X11

`.xfm`
: Verzeichnis mit Konfigurationsdateien des X-Filemanagers

Und viele anderen mehr...

Wenn es gewünscht ist, mehrere verschiedene Musteruser anzulegen, dann können neben dem skel-Verzeichnis

noch weitere solcher Verzeichnisse angelegt werden (z.B. skel_verwaltung, skel_produktion) und beim Anlegen der jeweiligen User muß dann useradd mit dem Parameter **-m -k *Verzeichnis*** aufgerufen werden, wobei *Verzeichnis* das gewünschte Musterverzeichnis ist.

1.111.3 - Konfigurieren und Nutzen der Systemlogs im Rahmen der administrativen und Sicherheitsanforderungen

Beschreibung: Prüfungskandidaten sollten in der Lage sein, die Systemaufzeichnungen zu konfigurieren. Dieses Lernziel beinhaltet das Einstellen der Art und Menge der aufgezeichneten Informationen, das manuelle Prüfen von Logdateien auf wichtige Aktivitäten, das Überwachen der Logdateien, das Einrichten automatischer Rotation und Archivierung der Logs und das Verfolgen von Problemen, die in den Logdateien aufgezeichnet

Die wichtigsten Dateien, Bezeichnungen und Anwendungen:

- **logrotate**
- **tail -f**
- `/etc/syslog.conf`
- `/var/log/*`

Unter Linux verwaltet ein spezieller Daemon-Prozess ein Systemlogbuch, das von allen Programmen und insbesondere von allen anderen Daemon-Prozessen benutzt werden kann, um Meldungen abzugeben. Dabei kann es sich um Meldungen des normalen Systemablaufs handeln oder um Alarmmeldungen, die sofortiges Eingreifen notwendig machen. Man spricht hier von unterschiedlichen **Prioritäten**.

Jedes Programm kann also über einen Systemaufruf solche Meldungen abgeben. Der Daemon, der diese Meldungen entgegennimmt und entscheidet, was mit ihnen zu geschehen hat heißt **syslogd**. Damit der Systemverwalter entscheiden kann, was mit welchen Meldungen geschehen soll kann dieser Daemon über die Datei **/etc/syslog.conf** konfiguriert werden.

In der Regel werden die verschiedenen Meldungen In Dateien geschrieben, die im Verzeichnis `/var/log` oder einem darinliegenden Unterverzeichnis abgelegt werden. Aber genau diese Frage, wohin wird welche Meldung geschrieben, wird in der Konfigurationsdatei ja eingestellt.

Die Datei /etc/syslog.conf

Jede Zeile dieser Datei, die nicht leer ist oder mit einem # beginnt, beschreibt eine Regel, was mit einer bestimmten Meldung passieren soll. die grundlegende Form jeder Zeile ist immer gleich und lautet:

```
Herkunft.Priorität     Aktion
```

Sollen in einer Regel mehrere Herkunftskategorien angegeben werden, so können sie, durch Kommas voneinander getrennt, nacheinander aufgeführt werden. Sollen mehrere Herkunfts/Prioritäts-Paare gleichzeitig angegeben werden, so müssen diese durch Strichpunkt getrennt angegeben werden. Sowohl Herkunft, als auch Priorität können durch ein Sternchen (*) ersetzt werden, das als Wildcard dient. In diesem Fall sind alle Herkunfts- bzw. Prioritätskategorien angesprochen.

Für Herkunft, Priorität und Aktion sind die folgenden Werte gültig:

Herkunftskategorien

kern	Systemmeldungen direkt vom Kernel
auth	Meldungen vom Sicherheitsdienst des Systems (login, ...)
authpriv	Vertrauliche Meldungen der internen Sicherheitsdienste
mail	Meldungen des Mail-Systems
news	Meldungen des News-Systems
uucp	Meldungen des UUCP-Systems
lpr	Meldungen des Druckerdaemons

cron	Meldungen des Cron-Daemons
syslog	Meldungen des syslog-Daemons selbst
daemon	Meldungen aller anderer Daemon-Prozesse
user	Meldungen aus normalen Anwenderprogrammen
local0-local7	frei verwendbar

Prioritäten in absteigender Reihenfolge

emerg	Der letzte Spruch vor dem Absturz
alert	Alarmierende Nachricht, die sofortiges Eingreifen erforderlich macht
crit	Meldung über eine kritische Situation, die gerade nochmal gut gegangen ist
err	Fehlermeldungen aller Art aus dem laufenden Betrieb
warn	Warnungen aller Art aus dem laufenden Betrieb
notice	Dokumentation besonders bemerkenswerter Situationen im Rahmen des normalen Betriebs
info	Protokollierung des normalen Betriebsablaufes
debug	Mitteilungen interner Programmzustände bei der Fehlersuche
none	Ist keine Priorität im eigentlichen Sinn, sondern dient zum Ausschluß einzelner Herkünfte

Eine angegebene Priorität meint immer die genannte oder eine höhere. Wenn jedoch vor der Priorität ein Gleichheitszeichen (=) steht, so ist nur die genannte Priorität gemeint.

Aktionen

Eine Aktion ist immer eine Weiterleitung einer Nachricht. Es gibt vier verschiedene Arten, wie solche Nachrichten weitergeleitet werden können:

1. **Ausgabe der Nachricht in eine Datei.**
   Dazu muß als Aktion der Dateiname mit absolutem Pfad (mit führendem Slash) angegeben werden. Normalerweise wird nach jedem Schreibvorgang des Syslog-Daemons eine Synchronisation des Dateisystems durchgeführt, weil sonst evt. Nachrichten bei einem Absturz nicht mehr physikalisch in die Datei geschrieben werden. Das ist allerdings eine sehr zeitaufwendige Aktion, daher gibt es die Möglichkeit, diese Synchronisation zu übergehen. Dazu wird dem absoluten Pfadnamen ein Bindestrich (-) vorangestellt.
2. **Weiterleitung der Nachricht an einen Syslog-Daemon eines anderen Rechners im Netz.**
   Dazu muß als Aktion der Rechnername des Rechners angegeben werden, an dessen Syslog-Daemon die Messages geschickt werden sollen. Dem Rechnernamen muß ein Klammeraffe (@) vorangestellt werden. Damit der angesprochene Syslog-Daemon auf dem anderen Rechner auch die Meldungen annimmt, muß er mit der Kommandozeilenoption **-r** (remote) gestartet worden sein.
3. **Ausgabe der Nachricht auf den Bildschirm von bestimmten Usern**
   Durch die Nennung des Usernamens (oder einer durch Kommas getrennten Liste von Usernamen) wird die Nachricht auf dem Bildschirm dieser User angezeigt, sofern sie eingeloggt sind.
4. **Ausgabe der Nachricht auf den Bildschirm aller eingeloggten User**
   In diesem Fall steht einfach ein Sternchen (*) im Aktionsfeld.

Ein paar Beispiele

Wie können nun solche Regelzeilen aussehen? Nehmen wir an, wir wollen dafür sorgen, daß alle Meldungen des Systems in die Datei /var/log/messages geschrieben werden. Das erledigt die Zeile

```
*.*         /var/log/messages
```

Wollen wir aber die Meldungen des Login-Systems, die eventuell Passwörter in Klarschrift enthalten ausschließen, so können wir das mit der Priorität none erledigen:

```
*.*;authpriv.none         /var/log/messages
```

Wir können dafür sorgen, daß Meldungen des Kernels, ab der Priorität warn immer auf den Bildschirm von root

geschickt werden, sofern er eingeloggt ist. Falls der User foo eingeloggt ist, soll er die Meldungen auch bekommen:

```
kern.warn      root,foo
```

Auch Gerätedateien sind Dateien. Schreiben wir alle Meldungen des Kernels die mindestens die Priorität warn haben und alle Meldungen ab error aller anderen Kategorien auf die Konsole 10 (/dev/tty10). Die Meldungen von authpriv lassen wir aber wieder weg. Auf Konsole 10 können sie dann auf dem Bildschirm mitgelesen werden:

```
kern.warn;*.err;authpriv.none   /dev/tty10
```

Die nächsten drei Zeilen schreiben

- Nur die kritischen Meldungen des News-Systems in die Datei /var/log/news/news.crit
- Nur die Fehlermeldungen des Newssystems in die Datei /var/log/news/news.err
- die bemerkenswerten Meldungen des News Systems in die Datei /var/log/news/news.notice

Alle Dateien werden nicht sofort synchronisiert (-)

```
news.=crit                      -/var/log/news/news.crit
news.=err                       -/var/log/news/news.err
news.=notice                    -/var/log/news/news.notice
```

Wenn wir einen zentralen Rechner haben, nennen wir ihn admhost, der das Logbuch für alle Rechner im Netz mitschreiben soll, so müssten alle Rechner im Netz die Zeile

```
*.*;authpriv.none          @admhost
```

in der syslog.conf eingetragen haben. Der syslogd auf admhost müsste dazu mit der Option -r gestartet sein.

Durch eine durchdachte Einteilung kann sich ein Systemverwalter viel Arbeit ersparen. Es ist ja möglich, daß alle denkbaren Ereignisse in verschiedene Dateien geschrieben werden, die dann schnell und effizient durchsucht werden können. Werden z.B. alle Meldungen ab der Priorität error (oder für vorsichtige Systemverwalter besser warn) in eine spezielle Datei geschrieben, so kann der Systemverwalter diese Datei regelmäßig überprüfen und muß sich nicht lange durch einen Berg von alltäglichen Meldungen kämpfen, um kritische Meldungen zu entdecken.

Verfolgen von Logdateien in Echtzeit

Manchmal ist es notwendig, Logdateien während einer bestimmten Aktion zu beobachten, um bestimmte Meldungen zu verfolgen. Um eine solche Beobachtung in "Echtzeit" zu ermöglichen gibt es zwei verschiedene Möglichkeiten. Die erste und am häufigsten angewandte Methode wird mit dem Befehl tail realisiert. tail kennt den Parameter -f, der eben diese Fähigkeit zur Verfügung stellt. Der Befehl

```
tail -f /var/log/messages
```

zeigt die letzten 10 Zeilen der Datei `/var/log/messages` am Bildschirm an. Statt aber dann zur Eingabeaufforderung zurückzukehren verharrt tail (durch den Parameter -f - forever) und wartet, ob die Datei "wächst", also neue Zeilen angefügt bekommt. Sobald neue Zeilen in die Datei geschrieben werden, werden sie auch auf dem Bildschirm ausgegeben.

Somit kann die Datei also "beim Wachsen beobachtet werden", erst ein Strg-C beendet diese fortlaufende Beobachtung. Diese Fähigkeit ist natürlich ideal geeignet, Log-Dateien zu beobachten, während ein Fehler gesucht wird.

Die zweite angesprochene Möglichkeit bietet das Programm less, der uns wohlbekannte Pager. less kennt auch die Möglichkeit einer Datei beim Wachsen zuzusehen, dazu muß der Befehl **F** eingegeben werden. Auch dieser Modus muß mit einem Strg-C abgebrochen werden.

Automatische Rotation der Logbuchdateien

Logbuchdateien werden zwangsläufig ziemlich schnell wachsen. Im normalen Betriebsablauf eines Linux-Rechners, der nicht nur untätig in der Ecke steht können pro Tag problemlos 50 bis 100 Kilobyte Meldungen entstehen. Da bietet es sich doch an, diese Logbücher hin und wieder zu löschen oder zu archivieren.

Im professionellen Umfeld sollten Logdateien niemals gelöscht werden. Sie sollten besser archiviert und abgespeichert werden, am besten zusammen mit dem Datum der Speicherung oder noch besser mit dem Start- und Enddatum. Einmal archiviert kann eine solche Datei dann komprimiert werden, Programme wie gzip sind bestens geeignet, Textdateien wie Logbücher mit immer wiederkehrenden Textpassagen auf eine sehr geringe Größe zu bringen. Eine Logdatei kann so tatsächlich weniger als ein Zehntel ihrer ursprünglichen Größe bekommen.

Verschiedene Linux-Distributionen gehen hier verschiedene Wege. Aber es zeichnet sich ein Standard ab, der dazu geeignet ist, zu große Logbuchdateien regelmäßig zu komprimieren und optional auch regelmäßig per Mail zu verschicken. Das Programm, das diesen Job übernimmt heißt logrotate und wird regelmäßig von cron (siehe Kapitel 1.111.4) aufgerufen.

logrotate bezieht seine Informationen aus einer (oder mehrerer) Konfigurationsdateien, die ihm als Kommandozeilenparameter mitgegeben wurden. Das genaue Format dieser Dateien ist auf der Handbuchseite erklärt.

Vielleicht sollte der Begriff Rotation noch einmal genauer erklärt werden. Er stammt aus der Welt des Backup, in der meist mehrere Bänder für ein Backup verwendet werden. Wenn wir beispielsweise jeden Freitag ein Backup machen, dann benutzen wir nicht jedesmal das selbe Band, sondern wir haben vielleicht 5 Bänder. Nun "rotieren" wir diese Bänder, das heißt, das wir in der ersten Woche das erste Band benutzen, in der zweiten das zweite usw. In der sechsten Woche benutzen wir dann also wieder das erste Band. So stehen uns immer die letzten 5 Wochen zur Verfügung.

Genauso läuft es mit den Logdateien. Das Programm **logrotate** rotiert die Logdateien, das heißt die gegenwärtige Logdatei wird - wenn ein bestimmtes Kriterium (Größe, Tag, Woche, Monat) erfüllt ist - in eine andere Datei kopiert (und optional komprimiert). Die Orginal-Logdatei wird dann gelöscht und neu angelegt. Die Anzahl der Rotationen (analog zu der Anzahl der verwendeten Bänder aus dem Backup-Beispiel) ist einstellbar. Das würde bedeuten, daß nach beispielsweise fünf Rotationen die älteste Datei gelöscht wird und statt dessen die neue Rotationsdatei angelegt wird. So haben wir die Sicherheit, daß immer nur eine begrenzte Anzahl von Dateien (und eine begrenzte Menge Speicherplatz) verwendet wird.

Optional kann man es aber auch so einrichten, daß die Logdatei, bevor sie dann endgültig gelöscht wird, an eine anzugebende E-Mail-Adresse gemailt wird.

1.111.4 - Automatisieren von Systemadministrationsaufgaben durch Planen von zukünftig laufenden Jobs

Beschreibung: Prüfungskandidaten sollten in der Lage sein, **cron** oder **anacron** zu verwenden, um Prozesse in regelmäßigen Intervallen ausführen zu lassen, und **at** zu benutzen, um Jobs zu einer bestimmten Zeit auszuführen. Dies beinhaltet das Verwalten von **cron** und **at** Jobs und die Konfiguration von Benutzerzugang zu **cron** und **at** Diensten.

Die wichtigsten Dateien, Bezeichnungen und Anwendungen:

- **at**
- **atq**
- **atrm**
- **crontab**
- `/etc/anacrontab`
- `/etc/at.deny`
- `/etc/at.allow`
- `/etc/crontab`
- `/etc/cron.allow`
- `/etc/cron.deny`
- `/var/spool/cron/*`

Systemverwaltungsaufgaben bestehen zu einem großen Teil aus immer wiederkehrenden Routinejobs. Diese Routine kann durch Kommandos vereinfacht werden, die automatisch immer wieder in bestimmten Intervallen oder zu bestimmten Zeiten ausgeführt werden. Für diese Aufgabe existiert ein eigener Daemon, der **cron**-Daemon. Er kann auf verschiedene Weisen konfiguriert werden, die hier genauer beschrieben werden sollen.

Manche Aufgaben sind aber nicht regelmäßig auszuführen, sondern nur genau einmal, aber zu einem ganz bestimmten Zeitpunkt. Diese Aufgabe wird vom **at**-Daemon gelöst, der genau dazu gemacht ist.

Der cron-Daemon und seine Konfiguration

Der Cron-Daemon überwacht verschiedene Dateien und Verzeichnisse, in denen Anweisungen liegen, die in regelmäßigen Abständen ausgeführt werden sollen. Diese Anweisungen werden Cron-Tabellen oder eben **crontabs** genannt.

Cron läd beim Start all die Dateien und Verzeichnisse, die er überwacht in den Arbeitsspeicher und überprüft jede Minute einmal, ob darin Jobs enthalten sind, die in der aktuellen Minute ausgeführt werden sollen. Wenn ja, so führt er sie aus. Außerdem überprüft cron jede Minute, ob sich an den Dateien oder Verzeichnissen etwas geändert hat und - falls ja - übernimmt er diese Änderungen im Speicher.

Es gibt also mehrere Möglichkeiten, dem Cron-Daemon einen Job zur Ausführung zu übergeben. Die einzelnen Möglichkeiten sind:

User-Crontabs
: Jeder User des Systems kann eine eigene Crontab (also eine Datei mit Anweisungen für Cron) erstellen und bearbeiten. Die Jobs, die darin aufgeführt sind werden von Cron unter der Userkennung des Users ausgeführt, um dessen Crontab-Datei es sich handelt. Die User-Crontabs werden nicht direkt mit einem Editor verändert, sondern mit dem Befehl crontab(1).

Systemweite Crontab-Datei
: Im Verzeichnis /etc existiert eine Datei `crontab`, die ebenfalls Cronjobs definiert. Das Format dieser Datei unterscheidet sich etwas von dem der Usercrontabs - siehe weiter unten...

cron.d Verzeichnis

Im Verzeichnis /etc kann ein Verzeichnis cron.d existieren, das Dateien im selben Format wie /etc/crontab enthalten darf. Auch diese Dateien werden von cron überwacht und darin enthaltene Befehle werden ausgeführt.

cron.{hourly,daily,weekly,monthly} Verzeichnisse

Ebenfalls im /etc/verzeichnis können Verzeichnisse der Namen cron.hourly, cron.daily, cron.weekly und cron.monthly existieren. Diese Verzeichnisse enthalten **nicht** crontabs, sondern Shellscripts, die entsprechend stündlich, täglich, wöchentlich oder monatlich ausgeführt werden.

Das Format der User-Crontabs

Das Format der Crontabs der User ist etwas speziell, hier das Grundprinzip: Crontab-Einträge sind entweder Definitionen von Shellvariablen oder zeitgesteuerte Cron-Befehle. Leere Zeilen oder Zeilen, die mit einem # beginnen werden ignoriert. Das Kommentarzeichen (#) darf aber nur am Anfang der Zeile stehen mitten in einer Zeile wird es als Teil des Kommandos interpretiert. Die Definition einer Shellvariable hat immer die Form:

```
Name=Wert
```

Einige Variablen werden von Cron selbst gesetzt. `SHELL` wird auf `/bin/sh` gesetzt, `HOME` und `LOGNAME` werden den Informationen über den User aus der /etc/passwd entnommen. Die Variablen `SHELL` und `HOME` dürfen verändert werden, `LOGNAME` nicht.

Normalerweise schickt cron alle Ausgaben von ausgeführten Kommandos per Mail an den User, der das Cron-Kommando ausführen liess. Die Variable `MAILTO` kann diese Eigenschaft verändern. Wenn sie den Namen eines Users enthält, so werden ihm und nicht dem ursprünglichen Cron-User die Mail geschickt. Ist die Variable `MAILTO` definiert, aber leer, so wird gar keine Mail verschickt.

Alle anderen Zeilen der Crontabs, die also nicht Variablen definieren, beschreibt Kommandos, die zu bestimmten Zeiten ausgeführt werden sollen. Das grundlegende Format dieser Zeilen ist:

```
Minute  Stunde  Tag  Monat  Wochentag          Kommando
```

Die einzelnen Felder definieren also die Zeit, zu der das angegebene Kommando ausgeführt werden soll. Jedes der fünf Zeitfelder darf statt einem Wert auch ein Sternchen (`*`) enthalten, das als Jokerzeichen gilt und für jeden beliebigen Wert steht.

Die jeweiligen Zeitfelder dürfen folgende Werte aufnehmen:

Minute	0-59
Stunde	0-23
Tag	1-31
Monat	1-12
Wochentag	0-7 (0 und 7 bedeutet Sonntag)

Auch Bereichsangabe sind erlaubt. Bereiche werden mit Bindestrich definiert. So würde die Angabe `8-11` im Stundenfeld die Stunden 8, 9, 10 und 11 meinen.

Auch Listen sind erlaubt. Listen werden durch Kommata aufgebaut. So würde die Angabe `1,2,5` im Wochentagfeld die Tage Montag, Dienstag und Freitag meinen.

Auch Kombinationen von Listen und Bereichen sind möglich. Die Angabe `1,3,15-19,30` spricht also die Zahlen 1, 3, 15, 16, 17, 18, 19 und 30 an.

Im Zusammenhang mit Bereichen können sogar Schrittweiten angegeben werden. Die Angabe `0-23/2` im Stundenfeld meint also die Stunden 0-23 aber in der Schrittweite 2 Stunden. Also 0, 2, 4, 6, 8, 10, ... Um diesen Schritt zu vervollständigen kann z.B. die Angabe *alle zwei Stunden"* einfach mit `*/2` ausgedrückt werden. Gleiches gilt für alle anderen Zeitfelder.

Das Wochentagfeld und das Tag-Feld sind - wenn beide angegeben sind - aditiv gemeint. Die Angabe

```
* * 13 * 5       ...
```

meint also nicht alle Freitag der 13., sondern alle Freitage **und** alle 13. des Monats.

Die letzte Angabe in einer Crontab-Zeile ist der Befehl, der zur gegebenen Zeit ausgeführt werden soll. Er wird von /bin/sh oder der in der Variable `SHELL` angegebenen Shell ausgeführt.

Eine Beispiel-Crontab könnte also folgendermaßen aussehen - mit Kommentaren zum besseren Verständnis:

```
# Variablendefinition
SHELL=/bin/bash
PATH=/usr/bin:/bin

# Cronbefehle
# Das Programm foo wird täglich um 23 Uhr 56 ausgeführt
56 23 * * *         foo

# Das Programm backup wird jeden Freitag um 17 Uhr 30 ausgeführt
30 17 * * 5         backup

# Alle 2 Stunden zwischen 6 und 23 Uhr wird das Programm fetchmail
# ausgeführt
0 6-23/2 * * *      fetchmail

# Am ersten jedes Monats wird um 8 Uhr morgens das Programm bar ausgeführt
0 8 1 * *           bar
```

Wenn der User, dem der crontab gehört die UserID 0 hat, also der Systemverwalter ist, dann kann eine Crontab-Zeile mit einem Bindestrich beginnen. In diesem Fall wird cron die ausgeführte Aktion nicht an den Syslog-Daemon weiterleiten. In jedem anderen Fall wird jede Cron-Aktion ins Logbuch übernommen und kann dort im Nachhinein überprüft werden.

User Crontabs verwalten mit crontab(1)

Die Crontabs der User sind zwar im Dateisystem als einzelne Dateien vorhanden (/var/spool/cron/tabs/*Username*), können aber dort nicht direkt editiert werden, weil sie nicht dem User gehören. Damit ein User trotzdem eigene Jobs definieren kann, steht ihm das Programm crontab zur Verfügung.

Dieses Programm erlaubt einem User, die folgenden Aktionen:

Anlegen einer neuen Crontab-Datei

Wenn ein User mit seinem Lieblings-Editor eine Datei erstellt hat, die seine Crontab-Befehle im oben beschriebenen Format enthält, und der User bisher keine Crontab besaß oder die bisherige Crontab durch diese neue Datei ersetzen will, dann kann er diese Datei mit dem Befehl

```
crontab Dateiname
```

zu seinem Crontab machen.

Ansehen der aktuellen Crontab

Mit dem Befehl

```
crontab -l
```

kann sich ein User seine Crontab am Bildschirm anzeigen lassen.

Löschen der Crontab-Datei

Wenn ein User seine Crontab-Datei löschen will, kann er dazu den Befehl

```
crontab -r
```

benutzen. Die ganze Crontab-Datei des Users wird dadurch gelöscht - alle bisherigen Jobs sind damit automatisch auch gelöscht. Die Veränderung wird sofort aktiv.

Einträge in der Crontab editieren

Mit dem Befehl

```
crontab -e
```

kann ein User seine Crontab-Datei editieren. Dazu wird der Editor aufgerufen, der in der Umgebungsvariable `VISUAL` bzw. `EDITOR` definiert ist - meist also der vi. Alle Veränderungen, die vom User dort vorgenommen werden, werden sofort nach dem Sichern und Verlassen des Editors aktiv.

Der Systemverwalter kann dem Programm crontab mit der Option `-u` *`username`* auch noch den User mitgeben, dessen Eintrag bearbeitet werden soll. Er kann also die Einträge jedes Users erstellen, löschen, ansehen und editieren. Ein Normaluser darf selbstverständlich nur seine eigene Datei bearbeiten.

Das Format der Crontabs in /etc

Die Datei `/etc/crontab` und alle Dateien (beliebigen Namens) im Verzeichnis `/etc/cron.d` werden auch als Crontabs gewertet und vom Cron-Daemon überwacht und ausgeführt. Im Wesentlichen entspricht das Format genau dem der User-Crontabs, mit einer Ausnahme. Zwischen dem fünften Zeitfeld (Wochentag) und dem Befehl wird hier noch ein Username angegeben, unter dessen ID der Befehl ausgeführt werden soll. Das Format ist also:

`Minute Stunde Tag Monat Wochentag Username Kommando`

Alle anderen oben genannten Eigenschaften bleiben unverändert, also auch die Bereichs- und Listenangabe der Zeitfelder.

Diese Dateien dürfen nur vom Systemverwalter angelegt und editiert werden, er kann aber durch das zusätzliche Userfeld bestimmen, unter wessen UserID der Befehl ausgeführt werden soll.

Die Verzeichnisse in /etc

Neben der Datei /etc/crontab und dem Verzeichnis /etc/cron.d existieren in vielen Linux-Distributionen (RedHat, Debian, SuSE,...) auch noch die Verzeichnisse

```
/etc/cron.hourly
/etc/cron.daily
/etc/cron.weekly
/etc/cron.monthly
```

Diese Verzeichnisse enthalten **keine** Crontabs, sondern Programme, die entsprechend stündlich, täglich, wöchentlich oder monatlich einmal ausgeführt werden sollen. In der Regel handelt es sich bei diesen Programmen um Shellscripts, die dann wiederum bestimmte Programme aufrufen. Standardeinstellungen, wann diese Programme ausgeführt werden sind:

- **cron.hourly** - Stündlich jeweils um *:30 Uhr
- **cron.daily** - Täglich jeweils um 0:00 Uhr
- **cron.weekly** - Wöchentlich jeweils Samstags um 0:00 Uhr
- **cron.monthly** - Monatlich jeweils am ersten eines Monats um 0:00 Uhr

In der Regel sind diese Verzeichnisse für regelmäßige Kommandos gemacht, die schon beim Installieren bestimmter Pakete festgelegt sind. Zumeist tragen die Scripts innerhalb dieser Verzeichnisse die Namen der

Pakete, die sie installiert haben.

Cron-Dienste für User erlauben oder verbieten

Normalerweise kann jeder User eines Linux-Systems seine eigene Crontab besitzen und beliebige Aufgaben zu beliebigen Zeiten damit erledigen lassen. Das kann er aber nur - wie oben gesehen - durch die Verwendung des Programms crontab. Dieses Programm bietet aber auch eine Art Zugriffsschutz, mit dem der Systemverwalter einzelnen Usern die Verwendung dieses Programms erlauben oder verbieten kann.

Wie so oft, gibt es hier zwei verschiedene Lösungsansätze. Entweder dürfen nur die User Crontab benutzen, die die ausdrückliche Erlaubnis besitzen, oder alle User dürfen crontab benutzen, denen es nicht explizit verboten wurde.

Dazu kann der Systemverwalter im Verzeichnis `/etc` mit zwei Dateien arbeiten, die er dort anlegen muß. Die folgenden Möglichkeiten existieren:

Existiert die Datei `/etc/cron.allow`, dann dürfen nur die User mit crontab arbeiten, die in dieser Datei aufgelistet sind.

Wenn die Datei `/etc/cron.allow` **nicht** existiert, stattdessen aber die Datei `/etc/cron.deny`, dann dürfen alle User mit crontab arbeiten, außer denen, die in der Datei `cron.deny` aufgelistet sind.

Beide Dateien sind reine Textdateien, die pro Zeile einen Usernamen enthalten dürfen.

In älteren Versionen von Cron lagen diese beiden Dateien nicht in /etc sondern hießen `/var/spool/cron/allow` und `/var/spool/cron/deny`. Die Funktionsweise war aber die selbe. Auf manchen älteren Linux-Versionen finden sich noch diese älteren Einstellungen. Nach der Revision hat LPI aber auf die neuen Dateien umgestellt.

anacron

Anacron ist wie cron ein periodischer Kommandoscheduler. Er führt bestimmte Befehle in Intervallen aus, die aber - im Gegensatzt zu cron - nur in Tagen angegeben werden. Der wesentliche Unterschied ist aber der, daß anacron nicht annimmt, daß der Rechner Tag und Nacht läuft.

Wenn cron einen bestimmten Job zu einer Zeit ausführen soll, zu der der Rechner nicht angeschalten ist, dann verfällt dieser Job. Oft ist es zum Beispiel so, daß die täglichen cron-Jobs um 0:00 Uhr ausgeführt werden sollen oder die wöchentlichen am Sonntag (Tag 0). Wenn ein Rechner aber weder Sonntags, noch Nachts läuft, werden diese Jobs niemals ausgeführt. Meist handelt es sich bei diesen Jobs um Verwaltungsaufgaben, wie der Auffrischung bestimmter Systemdatenbanken (locate und man) oder der Rotation der Logdateien.

Anacron kann dieses Problem lösen. Man kann einfach diese Jobs in Intervallen von einem, sieben oder 30 Tagen starten um tägliche, wöchentliche oder monatliche Ausführung zu erzwingen. Anacron führt seine Jobs aus, wenn der letzte Ablauf eines Jobs länger als die genannte Intervallzeit in Tagen her ist. Somit wird ein Job auch dann ausgeführt, wenn der Rechner das nächste Mal angeschalten wird und nicht - wie bei cron - wenn das nächste Wochen- oder Monatsende erreicht ist.

Jedesmal, wenn anacron aufgerufen wird, ließt es seine Konfigurationsdatei (/etc/anacrontab) in der seine Jobs mit den Perioden eingestellt werden. Wenn ein Job die letzten *n* Tage nicht ausgeführt wurde, wobei *n* die angegebene Periode dieses Jobs ist, führt anacron ihn aus. Anacron erzeugt dann einen Eintrag in einer speziellen Zeitmarkendatei, die es für jeden Job erstellt, so daß es weiß, wann der Job zuletzt ausgeführt wurde. Wurden alle Kommandos ausgeführt, wird anacron beendet.

Anacron ist also kein Daemon, der die ganze Zeit läuft, sondern muß entweder über init-Scripts oder cron regelmäßig gestartet werden.

Die Konfigurationsdatei von anacron ist sehr einfach aufgebaut. Sie enthält entweder Variablenzuweisungen, die der Umgebung zugewiesen werden, in der der entsprechende Befehl ausgeführt werden soll, oder Befehlszeilen der Form

```
Periode   Verzögerung   Job-Identifikation   Kommando
```

Die Periode ist eine Zahl, die die Anzahl von Tagen angibt, die zwischen der letzten Ausführung des Jobs und der nächsten Ausführung mindestens vergangen sein müssen. Die Verzögerung ist ein Wert in Minuten, der angegeben wird, damit nicht alle anacron-Jobs gleichzeitig gestartet werden und so den Rechner unnötig strapazieren. Somit können die verschiedenen Jobs leicht zeitversetzt gestartet werden. Die Job-Identifikation ist ein beliebiges Wort, daß alle Zeichen außer Leerzeichen und Slashs enthalten darf. Mit Hilfe dieses Wortes wird der Dateiname der Zeitmarkendatei erstellt (das ist der Grund für die verbotenen Slashs). Am Ende der Zeile steht dann der auszuführende Befehl. Eine simple `/etc/anacrontab` Datei könnte also folgendermaßen aussehen:

```
SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

 1    1    updatedb    updatedb
 7   10    logrotate   logrotate /etc/logrotate.conf
31   15    tmpclean    rm /tmp/*
```

Jeden Tag, eine Minute nachdem anacron gestartet wurde, wird der Befehl **updatedb** ausgeführt. Die Zeitmarke wird in eine Datei heißt ebenfalls updatedb.

Alle 7 Tage jeweils 10 Minuten nach dem Start von anacron wird der Befehl **logrotate /etc/logrotate.conf** ausgeführt. Die Zeitmarkendatei heißt nur logrotate.

Alle 31 Tage werden alle Dateien im /tmp Verzeichnis gelöscht, die Zeitmarkendatei heißt tmpclean. Der Befehl wird 15 Minuten nach Aufruf von anacron gestartet.

Hat anacron all diese drei Befehle abgearbeitet, dann beendet sich das Programm wieder.

Das at-Spoolsystem

Wenn ein Kommando nicht regelmäßig ausgeführt werden soll, sondern nur zu einem ganz bestimmten Zeitpunkt, dann können wir das mit dem at-Kommando erledigen. At ist ein typisches Unix-Spoolsystem, das Aufträge entgegennimmt, in eine Warteschlange stellt und zur gegebenen Zeit dann ausführt. Die Architektur ist wie bei allen anderen Unix-Spoolsystemen ausgeführt:

- Der Befehl **at** gibt Aufträge in die Warteschlange
- Der Befehl **atq** zeigt alle Aufträge, die in der Warteschlange stehen.
- Der Befehl **atrm** löscht einzelne Aufträge aus der Warteschlange.
- Der Daemon **atd** arbeitet die Warteschlange ab. In älteren Versionen von Linux findet sich kein eigener Daemon, in diesem Fall wurde von cron jede Minute einmal das Programm **atrun** aufgerufen, das die Warteschlange nach Aufträgen durchsuchte, die im Augenblick ausgeführt werden sollten und - sofern gefunden - diese Jobs ausführte.

Um mit **at** einen Job in Auftrag zu geben, muß einfach nur der Befehl

```
at Zeit
```

aufgerufen werden. Danach kann über die Standard-Eingabe die gewümschte Befehlszeile eingegeben werden, die zu der Angegebenen *Zeit* ausgeführt werden soll. Um die Eingabe abzuschließen, wird in einer eigenen Zeile das Dateiendezeichen Strg-D eingegeben. Alternativ kann eine Datei angelegt werden, die die Befehle enthält, die zu der bestimmten Zeit ausgeführt werden sollen. Dann kann **at** entweder mit

```
at Zeit < Datei
```

oder mit

```
at -f Datei Zeit
```

aufgerufen werden.

Die verschiedenen Formen der Zeitangaben enthehmen Sie bitte der Handbuchseite.

Die Ausgaben der Kommandos, die von **at** zu bestimmten Zeiten ausgeführt wurden, werden dem User als Mail zugeschickt, der den Auftrag losgeschickt hatte.

Wie schon bei cron, so können wir auch bei at festlegen, wer mit diesem Programm arbeiten darf bzw. wer nicht. Dazu dienen die Dateien `/etc/at.allow` und `/etc/at.deny`. Existiert die Datei /etc/at.allow, so dürfen nur die User mit **at** arbeiten, die dort aufgeführt sind (ein User pro Zeile). Existiert diese Datei nicht, aber die Datei /etc/at.deny, so dürfen alle User **at** benutzen, außer den Usern, die in at.deny aufgelistet sind. Existieren beide Dateien nicht, so darf nur der Systemverwalter mit **at** arbeiten.

1.111.5 - Aufrechterhaltung einer effektiven Datensicherungsstrategie

Beschreibung: Prüfungskandidaten sollten in der Lage sein, eine Sicherungsstrategie zu planen und Dateisysteme automatisch auf verschiedene Medien zu sichern. Die Tätigkeiten beinhalten das Sichern einer rohen Partition in eine Datei oder umgekehrt, das Durchführen teilweiser und manueller Backups, das Überprüfuen der Integrität von Backupdateien und teilweises oder vollständiges Wiederherstellen von Backups.

Die wichtigsten Dateien, Bezeichnungen und Anwendungen:

- **cpio**
- **dd**
- **dump**
- **restore**
- **tar**

Dieser Abschnitt der LPI102 Prüfung erfordert nicht nur das Wissen um die konkrete Syntax einzelner Befehle, sondern auch die grundsätzliche Architektur von Backup-Strategien. Backups werden heute mit den unterschiedlichsten Programmen erstellt und auf unterschiedlichsten Medien abgelegt. Die genaue Syntax aller hier denkbarer Programme ist sicherlich etwas, was den Rahmen sowohl der LPI-Prüfung, als auch dieser Darstellung sprengen würde. Die wichtigsten Bordmittel werden aber hier genau besprochen.

Vorüberlegungen

Grunsätzlich ist zu sagen, daß ein Backup auf jedem Rechner notwendig ist, dessen Dateien nicht rein statisch sind. Ein Rechner, der etwa nur als Router ins Internet Dienst tut, muß sicherlich kein regelmäßiges Backup ausführen, hier würde die einmalige Sicherung nach der abschließenden Konfiguration ausreichen. Jeder Rechner, auf dem gearbeitet wird, dessen Dateien also nicht statisch sind, muß aber regelmäßige Backups erledigen, um die getane Arbeit zu sichern. Wie im einzelnen die Backups aussehen, was gesichert wird, das muß gründlich geplant werden.

Ein wesentliches Kriterium für ein Backup ist die Frage, auf welches Medium die gesicherten Daten gespeichert werden sollen. Es bietet sich grundsätzlich an, Wechselmedien zu benutzen, die an einem anderen Ort gelagert werden können. Wird ein Backup nur auf eine zweite Platte im Rechner gemacht, so haben wir vielleicht eine Absicherung gegen eine physikalische Zerstörung der Platte (Headcrash), Feuerschäden, Wasserschäden oder Diebstahl des Rechners würde diese Form des Backups nicht abdecken, die Daten wären verloren.

Wechselmedien stehen in verschiedenen Formen zur Verfügung. Neben Wechselplatten, die auf der Festplattentechnik beruhen, sind es meist Bandlaufwerke, die für Backups verwendet werden. Hier spielen drei wesentliche Familien eine Rolle, Viertelzoll-Cartridges (QIC), DAT-Cassetten und Videobänder.

Die Speicherkapazität eines Backup-Mediums ist von großer Bedeutung. Muß das Medium während des Backups gewechselt werden, weil es nicht genügend Speicherplatz aufweist, so kann ein Backup nicht vollständig automatisiert werden. Der Aufwand (und damit auch die Hemmschwelle) ist wesentlich größer, als bei einem Medium, das das ganze Backup in einem Rutsch aufnehmen kann.

Die Frage, was überhaupt gesichert werden muß, ist die nächste Überlegung. Wir unterscheiden zwischen verschiedenen Backup-Strategien. Insbesondere zwei Begriffe werden hier immer wieder fallen: *volles Backup* und *inkrementelles Backup*.

Ein Vollbackup ist ein Backup, das alle Daten, die gesichert werden müssen, in einem Arbeitsgang auf ein Medium speichert. Ein inkrementelles Backup speichert im Gegensatz dazu nur die Daten, die sich seit dem letzten Backup verändert haben. **Ein inkrementelles Backup ist nur in Zussammenarbeit mit einem Vollbackup möglich!**. Abhängig von der Sicherheitsanforderung kann z.B. überlegt werden, jeden Tag der Woche ein inkrementelles Backup vorzunehmen, und Freitags ein Vollbackup. Das bedingt aber, daß alle Medien, die diese Backups

aufgenommen haben, zu einer Wiederherstellung (restore) der Daten benötigt werden.

Nehmen wir ein Beispiel: Wir haben folgenden Backup-Plan:

- Montag - inkrementelles Backup - Band 1
- Dienstag - inkrementelles Backup - Band 2
- Mittwoch - inkrementelles Backup - Band 3
- Donnerstag - inkrementelles Backup - Band 4
- Freitag - volles Backup - Band 5

Es ist jetzt Mittwoch, wir haben einen Headcrash, der uns zwingt, alle Daten auf eine neue Platte aufzuspielen. Wie müssen wir vorgehen?

Die Reihenfolge des Wiederaufspielens ist hier wichtig. Wir brauchen:

- Das Band mit dem letzten Vollbackup (Band 5)
- Die Bänder der letzten inkrementellen Backups vom letzten Vollbackup bis zum Eintreten der Katastrophe. Also in unserem Beispiel die Bänder von Montag und Dienstag (Band 1 und 2)

Das Wiederaufspielen muß jetzt in genau der Reihenfolge passieren, wie das Backup selbst. Zunächst spielen wir die Daten des Vollbackups wieder auf die Platte, dann die des Montag-Bandes und erst dann die des Dienstag-Bandes. Nur so stellen wir sicher, daß wir nicht alte Versionen von Dateien aufspielen, die die neueren wieder überschreiben...

Das ist ein aufwendiges Verfahren, das in der Praxis selten vorkommt. Um variabel mit Kombinationen von vollen und inkrementellen Backups arbeiten zu können, werden sogenannte Backup-Level definiert. Dabei wird ein Vollbackup immer Level 0 genannt, ein inkrementelles Backup hat dann Levelnummern größer als 0. Jedes inkrementelle Backup, egal welche Levelnummer es trägt, sichert immer alle Daten, die sich seit dem letzten Backup mit einer Nummer niedriger verändert haben. Ein Beispiel:

- **Erster Montag des Monats**
  Level 0 (Vollbackup)
- **Jeder andere Montag**
  Level 1 (wöchentliches inkrementelles Backup relativ zu Level 0)
- **Dienstag**
  Level 2 (tägliches inkrementelles Backup relativ zu Level 1)
- **Mittwoch**
  Level 2 (tägliches inkrementelles Backup relativ zu Level 1)
- **Donnerstag**
  Level 2 (tägliches inkrementelles Backup relativ zu Level 1)
- **Freitag**
  Level 2 (tägliches inkrementelles Backup relativ zu Level 1)

In diesem Fall brauchen wir für ein Wiederherstellen (restore) der Daten immer nur das aktuellste Band von jedem Level. Da jeder Level über 0 immer alle Daten sichert, die seit dem letzten Backup des vorgehenden Levels verändert wurden hat z.B. das Donnerstagsband immer auch die Daten gespeichert, die am Mittwoch verändert wurden. Es speichert ja alle Daten, seit dem letzten Backup mit niedrigerer Nummer, also seit Montag.

Zum Wiederherstellen der Daten nach einer Katastrophe am Freitag brauchen wir also

- Das letzte Vollbackup (Level 0)
- Das letzte inkrementelle Wochenbackup (Level 1)
- Das letzte inkrementelle Tagesbackup vom Donnerstag (Level 2)

Das Wiederaufspielen der Daten erfolgt jetzt in genau dieser Reihenfolge; zuerst das Vollbackup, dann Level 1 und erst dann Level 2. Jetzt ist garantiert, daß immer die aktuellsten Versionen der Dateien wiederaufgespielt sind.

Planung von Backups

Bei der Planung einer Backup-Strategie müssen ein paar Fragen durchüberlegt werden, deren Beantwortung das Backup leichter planbar machen. Wenn wir schon bei der Planung der Architektur des ganzen Systems auf solche Kriterien achten, können wir uns das Leben wesentlich erleichtern.

Die wichtigste aller Fragen ist die, welche Dateien überhaupt gesichert werden müssen. Bei einer vernünftigen Planung eines Systems können einige Teilbereiche des Systems völlig statisch angelegt werden. Das kann seinen Ausdruck dann darin finden, daß ganze Partitionen Read-Only gemountet werden, so daß selbst versehentliche Änderungen nicht möglich sind. Ein Beispiel hierfür ist die Tatsache, daß das `/usr`-Verzeichnis geradezu dafür gemacht ist, nur lesend eingehängt zu werden. Solche Bereiche können wir einmal - direkt nach der Installation - sichern und müssen sie dann nie wieder in unsere Backup-Überlegungen aufnehmen...

Eine weitere Überlegung beim Aufbau des Systems ist die, wo die User eines Systems Daten ablegen dürfen und wo nicht. Gibt es hier strenge Richtlinien, die etwa festlegen, daß User ihre Daten grundsätzlich nur in ihren Home-Verzeichnissen ablegen dürfen und diese Homeverzeichnisse auf einer separaten Partition liegen, dann ist durch das Backup dieser einen Partition schon der wesentliche Datenbesatz gesichert. Die Daten der User sind auf alle Fälle nicht verloren.

Ein weiterer wichtiger Bereich, der immer Teil des Backups sein sollte, ist der Bereich der Systemkonfigurationsdateien in /etc. Diese Konfiguration ist vielleicht nicht so hochdynamisch wie der Userdatenbereich, aber er enthält doch auch einige Daten, die sich oft verändern können. Neue oder veränderte Passwörter, die gesammte Arbeit der Systemkonfiguration usw. Um zu vermeiden, daß ein zu großer Datenbestand gesichert werden muß, ist es auch möglich, ein Verzeichnis auf einem Dateisystem anzulegen, das regelmäßig im Backup aufgenommen wird (z.B. /home). Dann wird ein Shellscript erstellt, das alle wichtigen Systemkonfigurationen aus /etc dorthin kopiert. Etwas wie

```
#!/bin/bash

cp /etc/passwd /home/backup
cp /etc/shadow /home/backup
cp /etc/group /home/backup
cp /etc/gpasswd /home/backup
cp /etc/fstab /home/backup
...
```

Dieses Script sollte jetzt täglich von cron aufgerufen werden. Damit ist sichergestellt, daß die wichtigsten Konfigurationsdateien ins Backup aufgenommen sind.

Backup mit dump

Das Programm **dump** bietet alle Funktionalität, um entweder ganze Dateisysteme (Partitionen) oder einzelne Verzeichnisse in ein Backup aufzunehmen. Die ursprüngliche Version von dump kommt aus der BSD Unix Welt, das Linux-Dump Programm ist aber eine eigene Version, die speziell für das EXT2 Dateisystem geschrieben wurde. Es gibt heute aber auch schon Versionen für weitere Dateisystemtypen, insbesondere für das ReiserFS.

Dump untersucht Dateien eines Dateisystems und entscheidet, welche Dateien in ein Backup aufgenommen werden müssen. Diese Dateien werden dann auf das angegebene Backup-Medium geschrieben, das entweder eine Platte, ein Band oder eine Datei sein kann. (Einstellbar mit der **-f** Option)

Passt das Backup nicht auf das angegebene Medium, so fordert dump zu einem Medienwechsel auf, und verteilt das Backup auf mehrere solcher Medien.

Dump unterstützt 10 Backup-Level (0-9), wobei wie oben beschrieben das Level 0 ein Vollbackup meint und jedes weitere Level ein inkrementelles Backup relativ zum letzten Backup des nächst-niedrigeren Levels bedeutet. Die grundsätzliche Aufrufform von dump ist

```
dump [-Level] [-u] [-f Backupdatei] Dateisystem
```

oder

```
dump [-Level] [-u] [-f Backupdatei] Verzeichnis
```

Wobei meist also zuerst das Backup-Level angegeben wird (0-9), gefolgt von einem u (update /etc/dumpdates). Mit der Anweisung -f *Backupdatei* wird das Medium gewählt. Hier steht also entweder eine Gerätedatei des verwendeten Bandlaufwerks oder ein Dateiname, der Datei, die das Backup aufnehmen soll. Die abschließende Angabe des Dateisystems oder Verzeichnisses gibt an, was gesichert werden soll.

Passt das Backup nicht auf das angegebene Medium, so fordert dump zu einem Medienwechsel auf, und verteilt das Backup auf mehrere solcher Medien.

Dump unterstützt 10 Backup-Level (0-9), wobei wie oben beschrieben das Level 0 ein Vollbackup meint und jedes weitere Level ein inkrementelles Backup relativ zum letzten Backup des nächst-niedrigeren Levels bedeutet. Die grundsätzliche Aufrufform von dump ist

```
dump [-Level] [-ua] [-f Backupdatei] Dateisystem
```

oder

```
dump [-Level] [-ua] [-f Backupdatei] Verzeichnis
```

Wobei meist also zuerst das Backup-Level angegeben wird (0-9), gefolgt von einem u (update /etc/dumpdates) und einem a, das die automatische Bandendeerkennung aktiviert. Dieses -a Argument ist bei modernen Bandlaufwerken besser als die Kalkulation, wieviel Daten auf das Band passen und wann es gewechselt werden muß. Bei der Verwendung von Dateien als Medium ist es unabdingbar diese Option zu setzen! Mit der Anweisung -f *Backupdatei* wird das Medium gewählt. Hier steht also entweder eine Gerätedatei des verwendeten Bandlaufwerks oder ein Dateiname, der Datei, die das Backup aufnehmen soll. Die abschließende Angabe des Dateisystems oder Verzeichnisses gibt an, was gesichert werden soll.

Die Datei /etc/dumpdates speichert die Daten alle bisherigen Backups in der Form:

```
/dev/hda8 0 Sat May  5 21:56:49 2001
/dev/hda8 1 Mon May  7 21:55:58 2001
...
```

Hier zeigt sich also, daß die Partition /dev/hda8 am Samstag, den 5. Mai 2001 um 21:48 Uhr ein Vollbackup (Level 0) erhalten hat, während am darauffolgenden Montag ein inkrementelles Backup (Level 1) ausgeführt wurde.

Der Aufruf

```
dump -0ua -f /dev/tape /dev/hda8
```

hatte das erste dieser beiden genannten Backups erstellt, das zweite wurde mit

```
dump -1ua -f /dev/tape /dev/hda8
```

erstellt.

Es sei hier an dieser Stelle noch einmal ausdrücklich an die Datei /etc/fstab erinnert, deren fünftes Feld die Frage klärt, ob die jeweilige Partition von dump bearbeitet werden soll, oder nicht.

Daten wiederherstellen mit restore

Das Gegenstück zum Programm dump ist **restore**, mit dem sich Daten von einem Backup-Medium wiederherstellen lassen. Dieses Programm hat sehr viele verschiedene Anwendungen, die alle mit Kommandozeilenparametern einstellbar sind. Die wichtigsten Aktionen für die wir dieses Programm brauchen sind nicht nur das Wiederherstellen von Daten, sondern auch der Vergleich eines Backups mit dem Orginaldatenbestand. Damit können wir bei wichtigen Backups die Konsistenz des Backups überprüfen.

Wichtige Aufrufformen von restore sind:

`restore -C -f *Datei*`

Das Backup, das sich in der *Datei*(Gerätedatei eines Tapes oder Datei) befindet wird mit dem Orginaldatenbestand verglichen. Eventuell auftretende Unstimmigkeiten werden angezeigt.

`restore -i -f *Datei*`

Das Backup, das sich in der *Datei*(Gerätedatei eines Tapes oder Datei) befindet wird mit einem interaktiven - shellähnlichen - Mechanismus durchwandert. Einzelne Dateien und Verzeichnisse können markiert werden und dann können alle markierten Dateien wiederhergestellt werden. Die wichtigsten Befehle dieser shellähnlichen Oberfläche sind:

cd *Verzeichnis*
: Wechselt in das angegebene Verzeichnis.

ls [*Verzeichnis*|*Datei*]
: Zeigt den Inhalt eines Verzeichnisses oder den Namen einer Datei. Ist eine Datei zur Wiederherstellung markiert, wird das durch ein * angezeigt.

add *Verzeichnis*|*Datei*
: Das angegebene Verzeichnis oder die angegebene Datei wird zur Wiederherstellung markiert. Ist es ein Verzeichnis so werden alle darin enthaltenen Dateien auch in die Liste der zu wiederherstellenden Dateien aufgenommen (wenn nicht die -h Option auf der Kommandozeile gesetzt war).

delete *Verzeichnis*|*Datei*
: Das angegebene Verzeichnis oder die angegebene Datei werden von der Liste der wiederherzustellenden Dateien gestrichen.

extract
: Die Dateien und Verzeichnisse, die in der Liste der wiederherzustellenden Dateien aufgelistet sind werden wiederhergestellt.

quit
: Das Programm wird beendet.

Zu beachten ist, daß restore beim Widerherstellen die Dateien im aktuellen Verzeichnis auspackt. Sollen also Dateien eines Dateisystems /dev/XXX wiederhergestellt werden, so bietet es sich an, auf die Wurzel dieser Partition zu wechseln.

`restore -r -f *Datei*`

Ein komplettes Dateisystem wird wiederhergestellt. Idealerweise ist das Ziel eine leere, frische Partition, in die dann gewechselt wird um den restore-Befehl auszuführen. Also z.B.:

```
mke2fs /dev/sda1
mount /dev/sda1 /mnt
cd /mnt

restore -r -f /dev/tape
```

Das ist die beste Lösung nach einem Plattencrash, wenn die orginale Platte zerstört ist und eine neue Platte komplett restauriert werden muß.

`restore -x -f *BackupDatei Datei1 Datei2 ...*`

Die angegebenen Dateien (*Datei1 Datei2 ...*) werden von dem Backup der Backupdatei (Gerätedatei oder Datei) gelesen und extrahiert. Auch hier wird wieder ins aktuelle Verzeichnis extrahiert. Sind einzelne der angegebenen zu extrahierenden Dateien Verzeichnisse, dann werden diese Verzeichnisse samt aller Dateien darin wiederhergestellt.

Restore ist also das Programm, mit dem alle Bearbeitung der bereits gemachten Backups vorgenommen werden, insbesondere die Wiederherstellung der gesicherten Dateien.

Ist ein Backup auf mehrere Medien verteilt, so fordert restore zum gegebenen Zeitpunkt zum Medienwechsel auf..

Partielle und manuelle Backups mit tar

Um schnell ein Verzeichnis oder ein paar Dateien zu sichern ist die Anwendung von dump und restore sicherlich übertrieben. Hier findet das Programm tar seine Anwendung. tar steht für Tape Archiver, ist also grundsätzlich auch gedacht gewesen, Backups auf Bandlaufwerke zu schreiben. Heute wird tar aber sehr häufig eingesetzt, um Verzeichnisse oder Dateien in eine Archivdatei zu packen, die dann auf anderen Medien abgespeichert werden kann oder auch an andere User weitergegeben werden kann. Vor der Einführung der modernen Paket-Manager (RPM, DPKG) war das tar-Format das bestimmende Format für Linux-Distributionen.

Tar wird sowohl zum Erstellen, als auch zum Entpacken von Archiven benutzt. Es ist also kein zusätzlicher Entpacker wie etwa restore nötig.

Wie schon dump und restore kennt auch tar die Option **-f *Dateiname*** mit der angegeben wird, welche Datei als Medium zum Speichern (oder entpacken) des Archivs verwendet werden soll. Auch hier kann entweder eine Gerätedatei eines Bandlaufwerks, eine reguläre Datei oder ein Bindestrich (für StdIn bzw StdOut) angegeben werden.

Tar selbst nimmt keinerlei Komprimierung vor, arbeitet jedoch gerne mit dem gzip-Programm zusammen. Musste man früher die Archivdatei noch manuell komprimieren, so kennt tar heute die Optionen -z (komprimieren oder dekomprimieren mit gzip) und -Z (komprimieren oder dekomprimieren mit compress - veraltet) um selbstständig gleich gepackte Archive zu erstellen oder zu entpacken. Die jeweiligen Programme gzip und compress müssen dazu aber vorhanden sein, tar ruft sie nur auf, es kann selbst keine Kompression vornehmen.

Die normale Endung einer Archivdatei, die mit tar hergestellt wurde heisst in der Regel `.tar` - wurde das Archiv mit gzip komprimiert trägt es standardmäßig die Endung `.tar.gz` oder (um auch DOS-konform zu sein) `.tgz`. Wurde die Archivdatei mit compress komprimiert, so endet ihr Name meist auf `.tar.Z`

Die drei wichtigsten Funktionen von tar sind

- Erstellen (create) von Archiven (-c)
- Inhaltsverzeichnis (table of content) von Archiven anzeigen (-t)
- Archive entpacken (extract) (-x)

Um ein Archiv mit dem Namen `Test.tgz` anzulegen, das mit gzip komprimiert ist und alle Dateien des Verzeichnisses `Testdir` enthält, rufen wir tar folgendermaßen auf:

```
tar -czf Test.tgz Testdir
```

Um den Inhalt dieses frisch geschaffenen Archivs anzusehen schreiben wir:

```
tar -tzf Test.tgz
```

Und um das Archiv im aktuellen Verzeichnis wieder zu entpacken ist der Befehl

```
tar -xzf Test.tgz
```

nötig. Wird zusätzlich noch die Option -v angegeben, so arbeitet tar "geschwätzig" (verbose), gibt uns also mehr Informationen mit. Insbesondere beim Ansehen des Inhaltsverzeichnis ist das eine gute Idee. Aber vorsicht: Der erste Parameter von tar legt die Aktion fest (c oder t oder x), die Parameter v und z folgen beliebig. Der f-Parameter kommt am Schluß, denn ihm folgt der Dateiname.

Archive mit cpio bearbeiten

Anstelle von tar kann auch das Programm cpio verwendet werden, um Archive zu erstellen oder zu bearbeiten. cpio kann dabei verschiedene Formate - unter anderem auch das tar-Format - bearbeiten. Ein großer Vorteil von cpio ist der, daß die Liste der zu archivierenden Dateien nicht auf der Kommandozeile (wie bei tar) erwartet wird, sondern

auf der Standard-Eingabe. Dadurch ist es möglich, die Ergebnisse etwa des find-Befehls direkt an cpio weiterzupipen und so ein Archiv mit den Suchergebnissen aufzubauen.

cpio arbeitet in drei verschiedenen Modi:

- **copy-out**
  Dateien werden in ein Archiv geschrieben (out meint die Ausgabe in ein Archiv)
- **copy-in**
  Dateien werden aus einem Archiv extrahiert (in meint die Eingabe ins Dateisystem)
- **copy-pass**
  Dateien werden von einem Ort zu einem anderen kopiert, quasi eine Kombination aus copy-out und copy-in, ohne jedoch tatsächlich ein Archiv zu erstellen.

Durch die Verwendung unterschiedlicher Formate und die Fähigkeit die zu archivierenden Dateien sowohl aus einer Dateiliste, als auch von der Standard-Eingabe zu lesen, ist cpio wesentlich flexibler als tar. Allerdings ist es auch um einiges komplizierter anzuwenden.

Ein- und Ausgabe mit dd

Um Dateien beliebiger Art (in unserem Fall speziell Archivdateien) von und auf Gerätedateien zu kopieren, existiert das sehr nützliche (und sehr häufig verwendete) Programm dd. dd steht für DiskDump und ermöglicht es, Daten von Gerätedateien zu regulären Dateien zu kopieren oder umgekehrt. Es ist natürlich auch möglich, von Gerätedatei zu Gerätedatei zu kopieren. Die grundsätzliche Form ist dabei

```
dd if=Eingabedatei of=Ausgabedatei bs=Blockgröße count=Anzahl
```

Dieser Befehl kopiert *Anzahl* Blöcke der Größe *Blockgröße* aus der *Eingabedatei* in die *Ausgabedatei*. Dabei können sowoh Ein-, als auch Ausgabedatei entweder Gerätedateien oder reguläre Dateien sein. Wird die Angabe der Blockgröße weggelassen, so werden die "natürlichen" Blockgrößen der jeweiligen Geräte verwendet. Wird die Anzahl der zu lesenden Blöcke weggelassen, so werden alle Blöcke der Datei (oder des Gerätes) gelesen. So kann z.B ein komplettes Image einer Festplatte oder Partition (oder Diskette oder CDROM) in eine Datei durch den Befehl

```
dd if=Gerätedatei_des_Laufwerks of=Imagedateiname
```

erstellt werden. Mit dem umgekehrten Befehl

```
dd of=Gerätedatei_des_Laufwerks if=Imagedateiname
```

wird das entsprechende Image wieder auf das Gerät geschrieben.

Durch die Angabe von ibs= (Input-Blocksize) und obs= (Output-Blocksize) kann sogar noch zwischen verschiedenen Blockgrößen bei Ein- und Ausgabe unterschieden werden.

Zusätzlich kann dd auch noch Konvertierungen der zu kopierenden Blöcke vornehmen, wie etwa die Reihenfolge der Bytes in einem Datenwort oder Klein-Großschreibung.

Für die Backup-Technik ist dd besonders interessant, weil damit Archivdateien direkt auf Band geschrieben werden können oder von Bändern gelesen werden können.

1.111.6 - Verwalten der Systemzeit

Beschreibung: Prüfungskandidaten sollten in der Lage sein, die Systemzeit richtig zu verwalten und die Uhr über NTP zu synchronisieren. Dieses Lernziel beinhaltet das Setzen von Systemdatum und -zeit, das Setzen der BIOS-Uhr auf die korrekte Zeit in UTC, die Konfiguration der korrekten Zeitzone des Systems und die Konfiguration der automatischen Korrektur der Zeit auf die NTP-Uhr.

Die wichtigsten Dateien, Bezeichnungen und Anwendungen:

- **date**
- **hwclock**
- **ntpd**
- **ntpdate**
- `/usr/share/zoneinfo`
- `/etc/timezone`
- `/etc/localtime`
- `/etc/ntp.conf`
- `/etc/ntp.drift`

Linux arbeitet mit zwei verschiedenen Uhren, die beide eine große Rolle spielen, die Hardware-Uhr (oder auch RTC, RealTimeClock, CMOS-Uhr, BIOS Uhr) und die Systemzeit. Um mit Linux und der Uhrzeit korrekt umzugehen müssen diese beiden Systeme genau unterschieden werden.

Die Hardware-Uhr ist ein Uhrenbaustein, der völlig unabhängig vom verwendeten Betriebssystem arbeitet. Es handelt sich um eine batteriegepufferte Uhr, die also selbst dann arbeitet, wenn der Computer ausgeschaltet (und von der Stromversorgung getrennt) ist.

Die Systemuhr von Linux ist eine Software-Uhr, die ein Teil des Linux-Kernel ist und durch einen regelmäßigen Timer-Interrupt gesteuert wird. Diese Software-Uhr zählt die Sekunden, die seit dem 1. Januar 1970 um 0:00 Uhr vergangen sind.

Unter Linux zählt nur die Systemzeit, also die kernelgesteuerte Software-Uhr. Der einzige Punkt, wo die Hardware-Uhr ins Spiel kommt ist der Moment des Systemstarts, wo die Systemzeit gestellt werden muß. Hier wird die Hardware-Uhr gelesen und die Systemzeit entsprechend eingestellt. Danach wird die Hardware-Uhr nicht weiter benutzt bis zum nächsten Systemstart.

Es gibt im Wesentlichen zwei Programme, die benutzt werden um auf die beiden Uhren zuzugreifen:

- **date**
  Anzeigen und Stellen der Systemzeit
- **hwclock**
  Anzeigen und Stellen der Hardware-Uhr

Bei Unix/Linux-Systemen bestehen jetzt zwei grundsätzliche Möglichkeiten, wie die Uhr gestellt werden sollte. Zum einen können wir die Uhr einfach nach der lokalen Zeit stellen, zum anderen haben wir die Möglichkeit, die Hardware-Uhr (und damit auch die Systemzeit) nach der UTC (Greenwich Mean Time) zu stellen und uns anhand einer entsprechenden Zeitzoneninformation die lokale Zeit daraus ausrechnen zu lassen. Diese zweite Möglichkeit ist heute die, der im Allgemeinen der Vorzug gegeben wird. Auch die Lernziele der LPI102 Prüfung zielen auf diese zweite Möglichkeit ab, also werden wir uns entsprechend daran halten.

Der Vorgang ist eigentlich ganz einfach. Wir stellen die Hardware-Uhr manuell (über das BIOS-Setup) auf Greenwich-Zeit und weisen der Umgebungsvariable TZ (TimeZone) den Wert zu, der unsere lokalen Zeitzone entspricht. Das verbreitete Shellscript **tzselect** bietet eine Auswahlmöglichkeit für die zur Verfügung stehenden Zeitzonen. In einigermaßen modernen Linuxen besteht zudem die Möglichkeit, in der Datei `/etc/timezone` die aktuelle Zeitzone einzutragen.

Die moderne Standard-Library von Linux kennt noch eine genauere Methode. Im Verzeichnis `/usr/share/zoneinfo` befinden sich binäre Informationsdateien für jede Zeitzone der Welt (inkl. Antarktis). Im Verzeichnis `/etc` kann jetzt ein symbolischer Link mit Namen `localtime` angelegt werden, der auf die entsprechende Zonendatei zeigt, also z.B. für Deutschland

```
ln -s /usr/share/zoneinfo/Europe/Berlin /etc/localtime
```

Um das zu vereinfachen existiert auf vielen Linuxen ein Script mit Namen **tzconfig**, das genau die Zeitzone abfrägt und anschließend den Link erstellt.

Ist die Zeitzone richtig eingestellt, so verhalten sich alle Linux-Programme, die mit Uhrzeiten arbeiten, automatisch korrekt. Jetzt können wir z.B. auch unsere Hardware-Uhr oder die Systemzeit stellen und die Programme hwclock und date rechnen aus der angegebenen Lokalzeit und der Information der Zeitzone die Greenwich-Zeit aus und stellen die Uhren entsprechend.

Damit Linux jetzt weiß, daß unsere Hardware-Uhr (und die Systemuhr) auf UTC (Greenwich) eingestellt ist, benutzen wir wieder das Programm hwclock. In einem Init-Script sollte die Anweisung

```
hwclock --utc --hctosys
```

stehen. (Falls die Uhr aber auf lokaler Zeit steht sollte statt --utc eben --localtime eingetragen sein)

date

Der Befehl **date** ist der normale Weg, wie auf die Systemzeit von Unix zugegriffen werden kann. Er bezieht sich aber immer auf die Systemzeit und nicht auf die CMos-Uhr (RTC Real-Time-Clock).

Als einfachste Form der Anwendung kann date einfach ohne Parameter aufgerufen werden. Dann gibt er die Systemzeit in einem String der Form

```
Sat Aug 21 14:17:40 MEST 1999
```

auf die Standard-Ausgabe aus. Das ist aber, gerade für Nicht-Amerikaner, nicht immer die gewünschte Form. Aus diesem Grund bietet **date** auch die Möglichkeit, eine beliebige Ausgabeform für ein Datum zu erstellen.

Wenn **date** einen Parameter erhält, der mit einem Pluszeichen (+) beginnt, so interpretiert es die folgende Zeichenkette und ersetzt bestimmte Elemente darin mit den entsprechenden Werten der Systemzeit.

Diese interpretierten Elemente beginnen immer mit einem Prozentzeichen (%) dem ein Buchstabe folgt. So bedeutet etwa das %H die Stunden im 24 Stunden-Modus, %M die Minuten. Wenn wir also nur wissen wollen, wie spät es gerade ist, könnten wir schreiben:

```
date "+Es ist jetzt %H Uhr und %M Minuten"
```

Die Anführungszeichen sind nötig, weil in unserer Zeichenkette Leerzeichen enthalten sind. Die Ausgabe des Befehls wäre jetzt etwas in der Art:

```
Es ist jetzt 14 Uhr und 25 Minuten
```

Eine genaue Darstellung aller unterstützter Parameter ist in der Handbuchseite aufgeführt.

Der Systemverwalter kann mit dem Befehl **date** auch die Systemuhr stellen, allerdings wird nur die Systemzeit, nicht die CMos-Zeit verändert. Dazu gibt es zwei Möglichkeiten:

1. Dem Kommando **date** folgt ein Parameter in der Form MMDDhhmm, also eine Zeichenkette, deren erste zwei Zeichen die Monatszahl (01-12) enthält, die nächsten zwei den Tag des Monats (01-31), die nächsten die Stunden (00-23) und die letzten die Minuten (00-59). Optional kann die Zeichenkette auch noch eine zwei oder vierstellige Jahreszahl und danach, durch einen Punkt getrennt die Angabe der Sekunden (zweistellig)

enthalten. (MMDDhhmmYYYY.ss)
2. Das Kommando **date** wird mit dem Schalter **-s** und einem darauffolgendem Datum der Art "Sat Aug 21 14:17:40 MEST 1999" aufgerufen. Das macht in der Regel nur dann einen Sinn, wenn vorher das date-Programm benutzt wurde um diese Zeit abzuspeichern, oder wenn ein date eines Rechners benutzt wird, die Uhren anderer Rechner zu stellen.

Verlässlichere Uhren

Sowohl die Hardware-Uhr, als auch die kernelbasierte Systemuhr sind nicht besonders verlässlich. Zwar bietet die Systemuhr die Möglichkeit über ausgeklügelte Mechanismen wie die Datei `/etc/adjtime` die Genauigkeit der Zeit zu erhöhen, aber eine wirklich hundertprozentige Zeit ist das auch nicht. Gerade aber bei Servern, die monatelang laufen, summiert sich auch eine kleine Ungenauigkeit auf eine störende Größenordnung. Aus diesem Grund gibt es ein spezielles Protokoll, das eine Synchronisation der Zeit über das Netz ermöglicht: das Network Time Protocol (NTP).

Damit NTP aber vernünftig arbeiten kann, benötigt es eine verläßliche Quelle für die Uhrzeit. Ein NTP-Server wird in der Regel eine Funkuhr besitzen, die eine absolut genaue Uhrzeit für den Rechner zur Verfügung stellt. Diese genaue Zeit kann er jetzt an NTP-Clients weitergeben.

Der Haken an dieser Sache ist, daß Pakete, die über das Netz transportiert werden, auch eine bestimmte Zeit benötigen, von einem Rechner zum anderen zu kommen. In einem lokalen Netz ist dieser Wert vernachlässigbar, im Internet können schon einige Sekunden zusammenkommen, je nach aktueller Verbindungsqualität und der Anzahl der Router, die zwischen den beiden Kommunikationspartnern liegen.

NTP hat auch für diese Frage eine Lösung parat. In der Regel werden nicht nur ein, sondern mehrere Server im Internet als Zeitquelle benutzt. Zu jedem Server wird eine Verbindung aufgebaut, und der Client errechnet sich aktuell, wieviel Zeit eine solche Verbindung dauert. Dazu sendet er eine Anfrage und wartet auf die Antwort. Er misst die entsprechend vergangene Zeit und errechnet sich so ein Mittel der Dauer, die ein Paket vom NTP-Server zu ihm benötigt. So kann er - mit mehreren Servern - auf eine Genauigkeit im Millisekundenbereich kommen, was völlig ausreicht, da unsere Uhr als kleinste Einheit sowieso mit Sekunden arbeitet.

Grundsätzlich geht es in diesem Zusammenhang nicht um den Aufbau eines NTP-Servers, der mit einer Funkuhr ausgestattet ist, sondern um die Anbindung eines Clients an einen existierenden Server. Diese Aufgabe ist sehr einfach zu realisieren, es gibt aber zwei verschiedene Möglichkeiten, einer solchen Anbindung:

- Anbindung über einen ständig laufenden Daemon-Prozess mit **ntpd**
- Anbindung über einen cron-gesteuerten Aufruf von **ntpdate**

Beide Möglichkeiten arbeiten mit dem NTP-Protokoll, aber im ersten Fall arbeitet unser Rechner auch als Server, während er im zweiten Fall nur Client ist.

Zeitsynchronisation mit ntpdate

Wenn es nur gewünscht ist, eine regelmäßige Anpassung der Uhrzeit über einen bestehenden Zeitserver vorzunehmen, so genügt es, mit dem Programm **ntpdate** die Uhrzeit eines öffentlichen Zeitservers einzuholen. In diesem Fall kann unsere Systemzeit vollkommen verstellt sein, die Uhrzeit (und das Datum) werden einfach vom genannten Server übernommen. Der Aufruf lautet:

```
ntpdate Zeitserver
```

wobei der Zeitserver entweder über seinen Namen oder seine IP-Adresse angegeben werden kann. Eine Liste öffentlicher Zeitserver im Internet ist unter http://www.eecis.udel.edu/~mills/ntp/clock1.htm erhältlich.

Natürlich bietet es sich an, diesen Aufruf regelmäßig über cron vorzunehmen. Ein entsprechender Eintrag in /etc/crontab könnte also lauten:

```
12 * * * *   root  /usr/sbin/ntpdate ntp3.fau.de
```

Damit würde einmal stündlich (jeweils um 12 Minuten nach der vollen Stunde) die aktuelle Uhrzeit vom Server ntp3.

fau.de (einem öffentlichen Zeitserver der Universität Erlangen) geholt und die Systemzeit danach gestellt.

Zeitsynchronisation mit ntpd

Komplexer ist die Aufgabe, die Zeit über einen Daemon zu stellen. Der **ntpd** oder auch **xntpd** ermöglicht diese Form der Synchronisation. Der Daemon synchronisiert sich alle 5 Minuten mit einem oder mehreren Zeitservern oder einer lokalen Funkuhr. Seine Konfiguration erhält er über die Datei `/etc/ntp.conf`. In dieser Datei stehen im einfachsten Fall nur zwei Einträge:

```
server ntp3.fau.de
driftfile /etc/ntp.drift
```

Die erste Zeile bestimmt den verwendeten Zeitserver, von dem regelmäßig (alle 5 Minuten) die Zeit geholt werden soll. Die zweite Zeile bestimmt eine Datei, die **ntpd** selbst anlegt und in der die Abweichung der Systemzeit zur realen Zeit gespeichert wird. **ntpd** ermittelt über Stunden hinweg die Ungenauigkeit der lokalen Zeit und ermittelt einen Wert der Ungenauigkeit in PPM (Parts per Million), der als einfache Fließkommazahl in der Datei `/etc/ntp.drift` abgelegt wird. Bei jedem Neustart von ntpd kann dann mit diesem Wert gerechnet werden.

Bei der Verwendung von **ntpd** als Daemon haben wir jetzt neben einer genauen Uhrzeit die Möglichkeit, selbst als Zeitserver für lokale Clients aufzutreten.

Im Gegensatz zur Verwendung von **ntpdate** darf die Systemzeit unseres Rechners aber nie wirklich größere Abweichungen enthalten. Sobald unsere lokale Zeit mehr als 1000 Sekunden Verschiebung zur Zeit des Servers enthält, weigert sich **ntpd** die Zeit tatsächlich zu stellen, sondern fordert (in einer Syslog-Meldung) dazu auf, die Zeit manuell zu stellen.

Es ist nicht möglich, beide Techniken auf einmal zu betreiben. Ein Versuch, **ntpdate** aufzurufen, obwohl ein lokaler **ntpd**-Prozess läuft, endet mit der Fehlermeldung:

```
13 Aug 14:37:05 ntpdate[3209]: the NTP socket is in use, exiting
```

Beide Programme benutzen den selben Socket, können also nicht nebeneinander existieren. Es ist aber sehr wohl möglich, im entsprechenden init-Script, das **ntpd** starten soll, einen Aufruf von **ntpdate** vor dem Aufruf des Daemons zu veranlassen, um sicherzustellen, daß die Systemzeit keine zu große Abweichung aufweist.